PAULO SÉRGIO PRAMPERO

NOVAS ABORDAGENS PARA OTIMIZAÇÃO MULTIMODAL BASEADAS EM ENXAMES DE PARTÍCULAS E CLUSTERIZAÇÃO

CAMPINAS
2014

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

PAULO SÉRGIO PRAMPERO

NOVAS ABORDAGENS PARA OTIMIZAÇÃO MULTIMODAL BASEADAS EM ENXAMES DE PARTÍCULAS E CLUSTERIZAÇÃO

Orientador: Prof. Dr. Romis Ribeiro de Faissol Attux

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE
DEFENDIDA PELO ALUNO PAULO SÉRGIO PRAMPERO
E ORIENTADO PELO PROF. DR. ROMIS RIBEIRO DE FAISSOL ATTUX

_____

CAMPINAS
2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

P885n

Prampero, Paulo Sérgio, 1973-
Novas abordagens para otimização multimodal baseadas em enxames de partículas e clusterização / Paulo Sérgio Prampero. – Campinas, SP : [s.n.], 2014.

Orientador: Romis Ribeiro de Faissol Attux.
Tese (doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Otimização. 2. Heurística. 3. Computação natural. I. Attux, Romis Ribeiro de Faissol,1978-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: New approaches for multimodal optimization based on particle swarm and clustering
Palavras-chave em inglês:
Optimization
Heuristic
Natural computing
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora:
Romis Ribeiro de Faissol Attux [Orientador]
André Carlos Ponce de Leon Ferreira de Carvalho
Carmelo José Albanez Bastos Filho
Rafael Ferrari
Levy Boccato
Data de defesa: 13-02-2014
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Paulo Sérgio Prampero

Data da Defesa: 13 de fevereiro de 2014

Título da Tese: "Novas Abordagens para Otimização Multimodal Baseadas em Enxames de Partículas e Clusterização"

Prof. Dr. Romis Ribeiro de Faissol Attux (Presidente): _____

Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho: _____

Prof. Dr. Carmelo José Albanez Bastos Filho: _____

Dr. Rafael Ferrari: _____

Dr. Levy Boccato: _____

Dedico à minha esposa Alessandra e ao meu filho Pedro.

AGRADECIMENTOS

Agradeço antes de tudo, a DEUS pela vida que é sempre mais importante. Agradeço pela brisa, pelo vento, pelo sorriso, pelo acalento, agradeço o abraço, o afago, agradeço o Amor que não se paga, só se retribui.

Agradeço ao meu Orientador Prof. Dr. Romis Ribeiro de Faissol Attux, um doutor da ciência e um mestre de amor, que mostrou o caminho para me tornar um Ser melhor. Desses anos de convivência só tenho elogios.

Agradeço ao Prof. Dr. Fernando José Von Zuben que abriu as portas da Unicamp para mim, aceitou-me como aluno especial, um pequeno gesto para um pode significar a mudança de vida para o outro.

Agradeço à minha primeira professora Dona Esmairdes, que lá em Poloni, no milênio passado, pegava na minha mão e me ajudava a escrever, e de alguma forma despertou em mim o desejo de estudar que nunca mais cessou. Agradeço aqui a todos os professores que passaram pela minha vida e me deram um pouco de si.

Agradeço a todos os funcionários da Unicamp que sempre me trataram com muita educação e presteza. Agradeço especialmente ao Edson Sanches, que me ajudou em tudo que precisei com imensa dedicação, e a Carmen Aparecida Fonseca pela sua gentileza e amabilidade.

Agradeço as boas almas que passaram pela minha vida, e que de alguma forma me amaram sem esperar nada em troca, são tantas e tenho tanto a agradecer e a retribuir.

RESUMO

Os algoritmos de otimização podem possuir características de busca local, global e multimodal. Em termos simples, os algoritmos de busca local procuram refinar uma solução inicial encontrada, promovendo a sua movimentação para o ótimo local dominante em sua vizinhança. Já os algoritmos de busca global possuem estratégias para escapar de ótimos locais, e, desta forma conseguem movimentar seus pontos de forma mais ampla pelo espaço de busca. Por fim, algoritmos multimodais procuram encontrar os vários ótimos no espaço de busca.

Neste trabalho foi proposto um algoritmo de otimização multimodal, baseado em PSO, com duas inovações: o raio de repulsão com controlador de passo, e o armazenamento da região promissora. O algoritmo foi chamado de MPSO (*Magnetic Particle Swarm Optimization*), e possui apenas um parâmetro de configuração, o número de partículas. Ele apresentou resultados bastante interessantes nos experimentos comparativos realizados, tanto em funções *benchmark* quanto em uma aplicação prática.

A segunda proposta foi uma metodologia de busca baseada em clusterização. Nesta metodologia, a modificação proposta translada os centroides das posições de representantes dos *clusters*, obtidas originalmente pelos algoritmos de clusterização clássicos, para uma região promissora no cluster, ou seja, para uma região melhor avaliada pela função objetivo, que será maximizada ou minimizada. Desta forma, a metodologia toma a forma de um operador que pode ser utilizado como pré-processador de algoritmos de otimização clássicos, sendo capaz de introduzir diversidade na população inicial, o que, como mostram os experimentos realizados, melhorou os resultados obtidos. Outras possibilidades de aplicação da ideia também são expostas e avaliadas.

Palavras-chave: Otimização multimodal, busca global, MPSO, otimização por clusterização.

ABSTRACT

Optimization algorithms have local, global and multimodal search features. In simple terms, a local search algorithm refines an initial solution by moving it towards an optimal dominant place in its neighborhood. Global search algorithms have strategies to escape from local optima, and thus can move their points more broadly throughout the search space. Finally, multimodal algorithms seek to find several global optima in the search space.

In this thesis, we propose a multimodal optimization algorithm based on PSO with two innovations: the repulsion radius with step controller and the storage of promising search directions. The algorithm was named MPSO (Magnetic Particle Swarm Optimization) and has only one configuration parameter, the number of particles. It led to consistent results in benchmark-based comparative experiments and in a practical application.

The second proposal was a clustering-based optimization methodology. This proposal moves the centroids from a position representative of a cluster, originally obtained by classical clustering algorithms, to a promising region in terms of an objective function to be maximized or minimized . The methodology was implemented in the form of an operator used as a pre-processor to traditional optimization algorithms. It introduced diversity in the initial population, which, as shown by the experiments,led to significant performance improvements. Other possibilities of applying this new idea are also discussed in the text.

Keywords*: Multimodal optimization, global search, MPSO, clustering-based optimization.*

LISTA DE ABREVIATURAS E SIGLAS

AG – Algoritmo Genético

CF – *Convergence Factor*

CLONALG – *Clonal Selection Algorithm*

DE - *Differential Evolution*

ED – *Estimation of Distribution*

EDA – *Estimation of Distribution Algorithm*

EM - *Electromagnetism-like Heuristic*

FCM - *Fuzzy C-means*

Opt-aiNet – *Artificial Immune Network for Continuous Optimization*

PSO – *Particle Swarm Optimization*

MPSO – *Magnetic Particle Swarm Optimization*

SIA – Sistemas Imunológicos Artificiais

SIN – Sistema Imunológico Natural

SUMÁRIO

1 INTRODUÇÃO

Um processo de otimização consiste em uma busca pelo ótimo, ou seja, uma busca pela melhor solução segundo uma métrica. Tal métrica pode ser um modelo matemático composto por uma função objetivo, um conjunto de variáveis de decisão e um conjunto de restrições. Esta composição permite modelar uma grande variedade de problemas.

Existem problemas de larga escala, os quais possuem um número grande de variáveis de decisão, outros podem ser multiobjetivo, que lidam com uma composição de funções. Há problemas com ótimo dinâmico, que variam ao longo do tempo, e estático, para os quais o ótimo não muda. Já os unimodais possuem apenas um ponto de ótimo global, e os multimodais, vários pontos de ótimo global. Além desses, existem os problemas com restrições lineares e não-lineares, e também com restrições compostas por variáveis dependentes e independentes, discretas ou contínuas (Camponogara, 2006).

A variedade de problemas de otimização é grande, isto reflete uma vontade humana de melhorar, de progredir e de otimizar. Na busca por um ótimo, existe também uma grande quantidade de soluções propostas para otimização com vários enfoques e perspectivas, e, segundo (Wolpert & Macready, 1997), não há uma proposta melhor universal.

Entretanto, três condições são desejáveis para um algoritmo de otimização (Gomes, 2006). A primeira é a garantia de convergência, normalmente obtida em propostas com forte embasamento matemático, e em contrapartida, com limitações de aplicabilidade, pois a garantia é dada sob certas restrições de aplicação.

A segunda é o tempo de convergência compatível com a aplicação. A convergência no tempo infinito pode eventualmente inviabilizar o uso da resposta. A terceira é a convergência para a solução ótima: encontrar uma solução ótima pode ser fundamental para o algoritmo. Pode ser aceitável que o algoritmo apresente uma boa solução, sem garantir o ótimo, mas é, em geral, indispensável que apresente uma boa solução factível.

Diante do exposto, duas propostas foram apresentadas para problemas de otimização multimodal com variáveis independentes e contínuas. A primeira proposta tomou a forma de um algoritmo de otimização baseado em otimização por enxame de partículas, com alterações nos processos de movimentação e de interação entre as partículas visando promover significativo aumento de diversidade. A segunda proposta consiste de uma metodologia de busca baseada em cluste-

rização. O processo de clusterização encontra um representante para o grupo e a metodologia proposta altera o posicionamento do representante para uma região melhor avaliada segundo a função objetivo.

A metodologia utilizada nesta tese foi eminentemente experimental, e o conhecimento científico apresentado está pautado na análise estatística das inúmeras simulações realizadas. A hipótese foi a introdução e manutenção de diversidade e seu refinamento com base nas características do algoritmo proposto. A segunda hipótese foi a introdução de diversidade com base nas características da metodologia.

O Capítulo 2 desta tese refere-se ao estudo de algoritmos de otimização, e começa com a delimitação do problema tratado nesta tese. Então, dois métodos de otimização não linear são apresentados: o método do gradiente e o método de Newton. Depois, a discussão caminha por métodos heurísticos, métodos imunológicos e evolutivos, e termina nos métodos baseados em inteligência coletiva, mais especificamente em PSO (do inglês, *Particle Swarm Optimization*), base da primeira contribuição desta tese. A importância deste capítulo se dá por apresentar algoritmos de otimização com inspirações variadas e conhecimentos relevantes para a posterior análise do algoritmo proposto.

O Capítulo 3 refere-se à apresentação do algoritmo proposto – otimização por enxame de partículas magnéticas ou MPSO (do inglês, *Magnetic Particle Swarm Optimization*). Neste capítulo, a proposta é justificada, sendo o algoritmo proposto descrito e aplicado a diversos experimentos. Seu desempenho é comparado com os de algoritmos clássicos na área. As principais funções escolhidas para teste foram *benchmarks* de otimização bem estabelecidos (Suganthan, et al., 2005), que visam prover bases para a análise do algoritmo quanto a sua capacidade de convergência e de busca multimodal e quanto a seu desempenho em comparação com alguns algoritmos descritos na literatura e com correlação com a proposta.

O Capítulo 4 refere-se à proposta da metodologia de busca baseada em clusterização. O capítulo começa com uma discussão sobre clusterização e otimização, e depois propõe alterações no *k-means* e no *c-means* para funcionarem como algoritmos de otimização. A proposta é implementada como um pré-processador, e as análises são feitas sob a capacidade de este pré-processador introduzir diversidade em outros algoritmos de busca.

Por fim a conclusão desta tese é apresentada, seguida de propostas de continuidade nos trabalhos realizados.

1.1 TRABALHOS PUBLICADOS E SUBMETIDOS DURANTE O DOUTORADO

Artigos publicados em e submetidos a periódicos:

1. Prampero, P. S.; Boccatto, L. ; Attux, R. R. F. Otimização Multimodal Baseada em Clusterização. Revista Sinergia (IFSP Online) ISSN: 2177-45. Submetido em 15/12/2013.

2. Prampero, P. S.; Attux, R. R. F. Magnetic Particle Swarm Optimization. Journal of Artificial Intelligence and Soft Computing Research, v. 2, pp. 05, 2013.

Trabalhos completos publicados em anais de congresso:

3. Prampero, P. S. ; Attux, R. R. F. Magnetic Particle Swarm Optimization. In: IEEE Symposium on Swarm Intelligence, 2011, Paris. Proceedings of the IEEE Symposium on Swarm Intelligence, pp.30-36, 2011.

4. Prampero, P. S.; Attux, R. R. F. Magnetic Particle Swarm Optimization with Estimation of Distribution. In: IEEE Congress on Evolutionary Computation, 2011, New Orleans. Proceedings of the IEEE Congress on Evolutionary Computation, 2011.

5. Prampero, P. S. ; Boccato, L. ; Attux, R. R. F. Multimodal Optimization as a Clustering Task: General Formulation and Application in the Context of Particle Swarm. In: X Congresso Brasileiro de Inteligência Computacional, 2011, Fortaleza. Anais do X Congresso Brasileiro de Inteligência Computacional, pp. 1-7, 2011.

6. Costa Junior, N.; Prampero, P. S. ; Attux, R. R. F. ; Assad, M. M. N. Estimate of Biogas Generation in Landfills Using Natural Computation Techniques. In: XXI Congresso Brasileiro de Engenharia Mecânica, 2011, Natal. Anais do XXI COBEM, 2011.

Resumos expandidos publicados em anais de congresso:

7. Prampero, P. S. ; Boccato, L. ; Attux, R. R. F. . Otimização Multimodal Baseada em Clusterização. In: V Encontro dos Alunos e Docentes do Departamento de Eng. de Computação e Automação Industrial - FEEC/UNICAMP, 2012, Campinas. Anais do V Encontro dos Alunos e Docentes do Departamento de Eng. de Computação e Automação Industrial - FEEC/UNICAMP, pp. 22-25, 2012.

2 FUNDAMENTOS E CONCEITOS BÁSICOS

Neste capítulo, serão apresentados alguns conceitos fundamentais para o entendimento deste trabalho e para a exposição das contribuições originais nele contidas. O capítulo tem início com a apresentação do problema abordado e com a definição do seu escopo, e, em seguida, passa-se à discussão de algoritmos clássicos de otimização não linear, bem como de métodos heurísticos e evolutivos; por fim, faz-se uma explanação mais detalhada do PSO e de suas variações, uma temática relevante para a análise da primeira contribuição original desta tese, apresentada no capítulo 3.

2.1 O PROBLEMA

Um processo de otimização objetiva, de forma geral, ressaltar certos atributos desejáveis e inibir atributos não desejáveis no âmbito de uma tarefa (Price, Storn, & Lampinen, 2005). Para se resolver um problema deste tipo, é necessário construir um modelo matemático, tal como aquele dado como exemplo na equação (2.1):

$$\begin{aligned} & min\ F(\mathbf{x}), \mathbf{x} \in R^n \\ \text{s.a.} \quad & H_i(\mathbf{x}) \leq r_i, i = 1, \dots, m \end{aligned} \tag{2.1}$$

onde $F(\mathbf{x}): R^n \rightarrow R$ é a função objetivo, $\mathbf{x}$ é o vetor com as n variáveis de decisão, $H_i(\mathbf{x}): R^m \rightarrow R$ é um elemento do conjunto de m restrições, r_i é um elemento de um vetor formado por m constantes reais e n é a dimensão do problema. Pelo exposto, o problema típico de otimização possui três partes formadoras: a função objetivo $F(\mathbf{x})$, o conjunto de variáveis de decisão $\mathbf{x}$ e o conjunto de restrições (Camponogara, 2006).

A função objetivo é uma função das variáveis de decisão que mensura, avalia ou quantifica o efeito de uma combinação qualquer de seus valores. Um exemplo tradicional nesta área é a otimização dos lucros de uma fábrica (Goldbarg & Luna, 2005). Neste exemplo, a função objetivo pode ser construída de modo a expressar o lucro de uma fábrica, e, como variáveis de decisão,

é possível utilizar a quantidade de produtos que serão produzidos, a quantidade de matéria prima para a fabricação destes produtos e os diversos parâmetros que influenciam a produção.

Desta forma, as variáveis de decisão são os parâmetros cujos valores definem uma solução para o problema. Essas variáveis podem ser dependentes ou independentes. As dependentes possuem seu valor calculado a partir das variáveis independentes, ou seja, modificando-se a variável independente, alteram-se os valores de suas variáveis dependentes (Pelikan, Goldberg, & Lobo, 2002). No final do processo de otimização delineado pelo exemplo da fábrica, o dono saberá quanto produzir de cada produto para obter um lucro máximo de acordo com o modelo proposto.

Existem diversas maneiras de formular o mesmo problema, e, para cada uma, é preciso escolher o processo de otimização mais adequado. Price, Storn e Lampiem (Price, Storn, & Lampinen, 2005) definiram características importantes do problema de otimização, que podem inclusive servir de parâmetro para escolha da técnica mais adequada. Estratégias para a construção de um modelo matemático podem ser encontradas em (Goldbarg & Luna, 2005).

Ainda mantendo o exemplo da fábrica, a otimização poderia ter, em vez de um, dois objetivos: minimizar os custos e maximizar os ganhos. Assim, o problema se tornaria multiobjetivo, havendo dois objetivos no caso (Deb, 2004). Note que o lucro máximo pode não estar no ponto de custo mínimo e nem na posição do ganho máximo, mas na maior diferença entre o custo e o lucro. Em termos simples, um problema de otimização multiobjetivo se caracteriza por possuir um conjunto de funções objetivo a serem otimizadas (maximizadas ou minimizadas) (Deb, 2004).

O conjunto de restrições desempenha um papel importante na modelagem. Ainda no exemplo, ao minimizar a função custo sem restrições, a solução pode tender a zero, por exemplo, caso se enfoque apenas o custo de produção. Por outro lado, uma produção "sem limites" dos itens que fabrica também não seria realista, pois o mercado pode não absorver tal produção, reduzindo o preço do produto pelo excesso de oferta num cenário dinâmico.

Sendo assim, existe em muitos casos a necessidade de limitar / restringir a amplitude de variação das variáveis de decisão, e, por isto, ocorre a necessidade do conjunto $H(\mathbf{x})$[1]. Toda solução que satisfaz todas as restrições de $H(\mathbf{x})$ é dita factível (Gonçalves, 2011) . Factibilidade significa, em essência, que uma solução pertence ao domínio efetivo do problema.

[1] Um problema de otimização é dito irrestrito quando não possui o conjunto de restrições H(**x**).

2.1.1 Maximização e Minimização

Dependendo da natureza da tarefa em mãos, pode-se ter interesse em buscar os maiores ou menores valores de custo. O processo de *maximização* consiste em encontrar pontos que produzam o maior valor possível para a função objetivo. Já o processo de *minimização* consiste em encontrar pontos para os quais o valor da função objetivo seja o menor possível.

Existe uma dualidade no sentido de que maximizar $F(\mathbf{x})$ é equivalente a minimizar $-F(\mathbf{x})$ (Gonçalves, 2011). Isto significa que o maior valor de $F(\mathbf{x})$ é necessariamente o menor valor de $-F(\mathbf{x})$. Desse modo, pode-se pensar em ambos os casos sob a égide do conceito genérico de otimização.

Em termos simples, um processo de otimização tem como objetivo encontrar pontos de ótimo global dentro da região factível. A expressão "ótimo global" refere-se a um ponto de máximo global e/ou de mínimo global, dependendo do caso. Um ponto $\mathbf{x}^*$ é dito *mínimo global* de $F(\mathbf{x})$ se:

$$F(\mathbf{x}^*) \leq F(\mathbf{x}),\ \forall\, \mathbf{x}\ \in\ R^n \tag{2.2}$$

Um ponto $\mathbf{x}_L$ é ponto de *mínimo local* de $F(\mathbf{x})$ se existe $B(\mathbf{x}_L, \varepsilon)$, um conjunto ao redor $\mathbf{x}_L$, e um ε um valor real pequeno, tal que:

$$F(\mathbf{x}_L) \leq F(\mathbf{y}),\ \mathbf{x}_L\ \in\ R^n\ e\ \mathbf{y}\ \in B \tag{2.3}$$

Note que $\mathbf{x}^*$ é um ótimo relativamente a todo o domínio da função, e que $\mathbf{x}_L$ o é apenas em seu entorno.

Um ótimo global também é, naturalmente, ótimo local, vide Fig. 2.1 (Bazaraa, Shetty, & Sherali, 1993). Assim as referências aos mínimos globais excluem os mínimos locais, mas as referências aos mínimos locais valem para os mínimos globais.

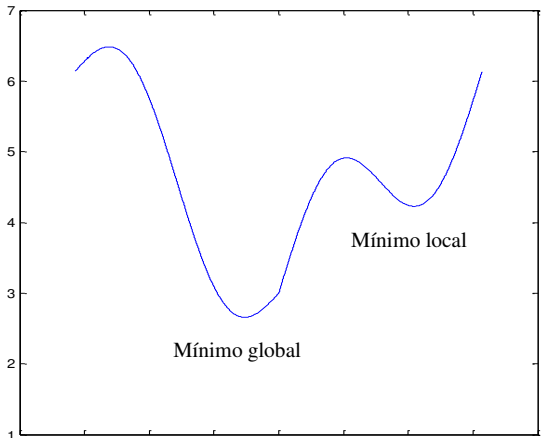


Figura 2.1: Mínimo local e mínimo global.

Um processo de otimização é realizado por métodos de busca, que são algoritmos que amostram o espaço de busca, o domínio efetivo da função. Esses métodos podem realizar busca local, global e multimodal.

O que caracteriza um comportamento de busca local é o refinamento, ou seja, a capacidade de melhoria de uma solução no âmbito de uma vizinhança. Uma solução poderia, por exemplo, sofrer uma pequena perturbação, idealmente suficiente para melhorar a avaliação do ponto pela função objetivo, sem, no entanto, retirá-la da bacia de atração do ótimo local onde se encontra.

É muito comum que um algoritmo de busca local tenha comportamento guloso, só alterando o ponto atual quando encontra um ponto com avaliação superior. Com este comportamento, o algoritmo tende mover a solução inicial encontrada para um ótimo local na região, desde que esta não esteja originalmente em um platô (Russell & Norvig, 2004).

Já a busca global requer, em geral, capacidade de escapar de ótimos locais, isto é, potencial de explorar novas regiões do espaço. Assim, um método de busca global aplica, eventualmente, variações maiores a uma solução. Uma estratégia clássica neste sentido é aceitar, mesmo que temporariamente, que um novo ponto adotado possua uma avaliação pior do que o atual (Russell & Norvig, 2004).

Uma função é dita *multimodal* quando possui mais de um máximo ou de um mínimo global no domínio estudado (Michalewicz, 1996). Um exemplo é dado na Fig. 2.2, que contém uma superfície com quatro ótimos globais e vários ótimos locais. Idealmente, um algoritmo destinado

a resolver problemas multimodais deve procurar encontrar todos os ótimos globais de uma função, e, para isto, manter um bom espalhamento de soluções no espaço de busca. A manutenção de soluções variadas ao longo das iterações de algoritmos populacionais é chamada de *manutenção da diversidade*.

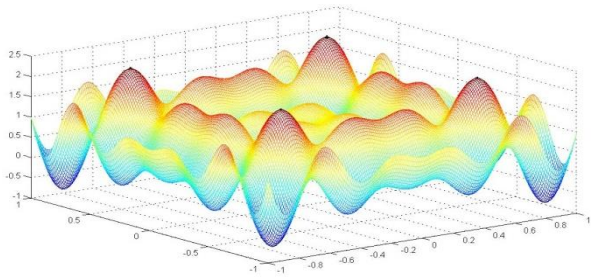


Figura 2.2: Exemplo de função multimodal.

Em suma, os algoritmos de busca local procuram refinar uma solução inicial encontrada, promovendo a sua movimentação para o ótimo local dominante em sua vizinhança. Já os algoritmos de busca global possuem estratégias para escapar de ótimos locais, e, desta forma conseguem movimentar seus pontos de forma mais ampla pelo espaço de busca. Por fim, algoritmos multimodais procuram encontrar os vários ótimos no espaço de busca.

Após o exposto, é possível definir o escopo de aplicação desta tese. O problema que será abordado é o de otimizar funções multimodais com um único objetivo, definidas em um espaço real contínuo com variáveis independentes dentro de um domínio limitado com restrições.

Nas próximas seções, serão apresentadas técnicas de busca local, global e multimodal que vão desde métodos baseados em derivadas da função objetivo até metaheurísticas populacionais bio-inspiradas.

2.2 MÉTODOS DE OTIMIZAÇÃO DE FUNÇÕES

Nesta seção, serão apresentados diversos métodos de otimização de funções definidas sobre um domínio contínuo. As técnicas aqui apresentadas, com pequenas variações, podem ser

aplicadas a outros contextos, como o de otimização combinatória, mas isto não será feito neste trabalho.

Os métodos descritos nesta seção, no contexto de otimização de funções, possuem nomenclaturas diferentes para os mesmos componentes. Por exemplo, o indíviduo, partícula, ou anticorpo é um agente reativo simples no processo de busca baseado em população, cuja posição representa uma possível solução. Além disso, a função objetivo também receberá nomes alternativos, como função de *fitness*.

Um método de otimização é dito determinístico quando não possui passos estocásticos. Assim, dado o mesmo conjunto de pontos iniciais, ao se aplicar um método determinístico obtém-se, ao final do processamento, a mesma resposta (Brandão, 2010). A seguir, discutiremos um conjunto de métodos determinísticos e estocásticos que é representativo do ponto de vista da temática deste trabalho.

2.2.1 Métodos de Otimização Não-Linear

Os métodos clássicos de otimização não-linear são determinísticos. A principal ferramenta utilizada nesses métodos é a informação trazida pelas derivadas da função objetivo o, que, por um lado, tipicamente garante a convergência do método para um ótimo local, embora, por outro lado, limite sua aplicabilidade, uma vez que a função objetivo deve ser contínua e diferenciável.

Uma forma de classificar os métodos de otimização não-linear é pela ordem da informação das derivadas. Nos métodos de primeira ordem, são utilizados valores da função objetivo e de suas derivadas em relação às variáveis de decisão. Dentre os métodos dessa classe, temos o método *steepest descent* e o método do gradiente conjugado (Silva, 2013). Já os métodos de segunda ordem empregam, adicionalmente, informações trazidas pela matriz hessiana. Exemplos desta classe são os métodos de *Newton* e *Quasi-Newton* (Luenberger, 1984).

Também podem ser mencionados os métodos de terceira ordem, como, o método de *Halley* (Brown Jr, 1977), que, sob certas condições, garante uma convergência cúbica, embora com um custo computacional muito elevado. A rapidez na convergência aumenta, em essência, com a ordem empregada (Silva, 2013).

Para ilustrar a discussão realizada, a seguir serão expostos dois métodos clássicos de otimização não-linear.

2.2.1.1 Método do Gradiente / *Steepest Descent*

No método do gradiente (também denominado método *steepest descent* no caso de minimização), conduz-se a busca de acordo com a direção e sentido do gradiente da função objetivo no ponto enfocado. Exige-se que a função seja diferenciável no ponto em questão, de modo que possam calcular as derivadas pertinentes.

O vetor gradiente é denotado por:

$$\nabla F(\mathbf{x}) = \begin{pmatrix} \frac{\partial F}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial F}{\partial x_n}(\mathbf{x}) \end{pmatrix} \tag{2.4}$$

Pode-se mostrar que esse vetor indica localmente a direção de maior crescimento da função (Oliveira, 2012). O sentido contrário ao do gradiente assinala o maior decrescimento, levando a uma direção minimizante:

$$d = -\frac{\nabla F(\mathbf{x})}{\|\nabla F(\mathbf{x})\|} \tag{2.5}$$

Obtida essa direção de minimização, resta determinar o tamanho do passo de adaptação para que se possa construir uma estratégia iterativa. Caso o passo seja muito pequeno, o processo será lento; caso seja muito grande, o processo pode se tornar dinamicamente instável (Oliveira, 2012).

Uma estratégia simples para abordar o problema de escolha do passo é atribuir a ele valor unitário, e mantê-lo com este valor enquanto as iterações levarem a melhoras no custo. Entretanto, sempre que ocorrer uma piora, deve-se diminuir o passo por um fator multiplicativo pré-determinado. Desta forma, a cada iteração, deverá ocorrer uma contínua aproximação do ótimo local. Um pseudocódigo possível é descrita no Alg. 2.1.

Algoritmo 2.1: Método do Gradiente

1) **Gerar** um ponto x em uma posição aleatória dentro região factível
2) **Calcular** a direção d_K no ponto x_k
3) **Calcular** $\nabla F(x_k)$ para o ponto x_k
4) **Calcular** ponto x_{k+1}

$$x_{k+1} = x_k + \propto_k d_k$$

5) **Se** $(F(x_{k+1}) \geq F(x_k))$ altere $\propto_k$.
6) **Se** o critério de parada não for satisfeito, **voltar ao passo 2**

Pode-se utilizar como critério de parada o tamanho do passo: quando este for considerado muito pequeno, o algoritmo termina, assumindo-se convergência local. Cabe ressaltar que, na escolha do passo, podem ser utilizados métodos mais rigorosos, como uma busca unidimensional baseada na seção áurea (Bazaraa, Shetty, & Sherali, 1993).

O método do gradiente utiliza informações de primeira ordem, e, devido à necessidade de definição exata do passo, o método pode apresentar convergência muito lenta, principalmente perto do ótimo local. Contudo, é importante citar que uma iteração do método do gradiente apresenta um esforço computacional relativamente baixo em comparação com métodos de segunda ordem, como o método de Newton, que será visto a seguir.

2.2.1.2 Método de Newton

O método de Newton é um método de segunda ordem, pois utiliza a matriz hessiana da função objetivo, representada por $\nabla^2 F(\mathbf{x})$, para encontrar um ponto de ótimo dessa função (Camponogara, 2006). A matriz hessiana é dada por:

$$\nabla^2 F(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 F}{\partial x_1 \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_n \partial x_n}(\mathbf{x}) \end{pmatrix} \qquad (2.6)$$

Essa matriz pode ser calculada quando a função for duas vezes diferenciável. Note ainda que a matriz hessiana tem como elementos as derivadas dos elementos do vetor gradiente.

Para que uma função $F(\mathbf{x})$ tenha seu máximo ou mínimo em um ponto $\mathbf{x}$, a condição primordial é que a derivada primeira neste ponto seja nula. A natureza desse ponto $\mathbf{x}$ pode ser avali-

ada pelos autovalores da matriz hessiana (Camponogara, 2006). Caso todos os autovalores da matriz hessiana em **x** sejam positivos, conclui-se que a função tem mínimo em **x**. Por outro lado, caso todos os autovalores da matriz hessiana sejam negativos, então a função tem máximo em $\boldsymbol{x}$. Caso haja autovalores positivos e negativos, será um ponto de sela da função.

Assim, pode-se conceber o método de Newton como um processo iterativo de busca por um ponto mínimo x_k, tal que $\nabla F(\mathbf{x}_k)$ seja igual a zero e $\nabla^2 F(\mathbf{x}_k)$ seja maior do que zero, ou seja, definida positiva (Brandão, 2010). Para aplicar o método a um problema de maximização, será preciso considerar um sinal oposto da matriz hessiana.

No caso de minimização, a lei de ajuste do método de Newton é (Oliveira, 2012):

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \propto_k M_k^{-1} \nabla F(\mathbf{x}_k) \qquad (2.7)$$

onde, levando-se em conta o processo de positivação da matriz hessiana, M_k é dada por:

$$M_k = \nabla^2 F(\mathbf{x}_k), \text{ se } \gamma_{min}^{[k]} > 0 \qquad (2.8)$$

$$M_k = \nabla^2 F(\mathbf{x}_k) + (\varepsilon - \gamma_{min}^{[k]})I, \text{ se } \gamma_{min}^{[k]} < 0 \qquad (2.9)$$

onde $\gamma_{\min}^{[k]}$ é o menor autovalor de M_k, $\varepsilon > 0$ é um valor pequeno arbitrário e $0 < \alpha_k \leq 1$. O valor de α_k pode ser escolhido de acordo com as diretrizes delineadas na seção 2.2.1.1.

O método de Newton tipicamente apresenta taxa de convergência maior que a do método do gradiente, mas, como já mencionado, é mais custoso, uma vez que precisa da inversão da matriz M.

2.2.2 Métodos Heurísticos

O termo "heurístico" é proveniente do grego, e, nessa língua, possui a mesma raiz do verbo "descobrir". Tecnicamente, um método heurístico é um método que aplica um conjunto de regras que permitem a descoberta de uma solução de um problema (Gomes, 2006).

De forma geral, os métodos heurísticos modificam soluções aplicando sobre elas regras construídas com o objetivo de obter soluções melhor avaliadas pelo critério adotado, ou seja, pela

função objetivo. Portanto, uma heurística é uma técnica de busca de aplicação ampla, que utiliza procedimentos gerais para encontrar o ótimo de uma função, sem que necessariamente se tenha conhecimento prévio da mesma.

A grande vantagem do ponto de vista de aplicabilidade é que um método heurístico não requer a satisfação de complexas condições *a priori* acerca das funções, pois apenas utiliza a função objetivo para avaliar pontos. Um aspecto que se deve ter em conta é que não há, via de regra, garantias teóricas de convergência para um ótimo global dentro de um limite finito de tempo.

2.2.2.1 Hill Climbing

Hill climbing, ou subida da montanha, como o próprio nome diz, é um método de busca local que utiliza um procedimento de melhora iterativa de uma solução candidata inicialmente gerada (Castro L. N., 2006). O *hill climbing* padrão começa gerando uma solução candidata dentro do espaço de busca factível e provoca pequenas perturbações nesta solução. No caso de melhoria, a solução modificada passa a ser a nova solução e o processo continua até que não aconteçam melhorias por várias iterações ou até que um limite de iterações seja atingido. Desta forma, o algoritmo apenas altera sua solução quando a nova solução for melhor que a antiga, o que o caracteriza como um método guloso. Uma descrição geral está no Alg. 2.2.

Algoritmo 2.2: *Hill Climbing*

```
1) Gerar um ponto x em uma posição aleatória dentro da região factível
2) Em cada iteração, gerar um novo ponto x', obtido através da aplicação
de uma pequena perturbação no ponto x
3) Se x' tiver melhor valor junto à função objetivo, então ele se torna o
ponto atual
```

O processo de provocar pequenas modificações no ponto atual torna o *hill climbing* um método estritamente local, uma vez que as perturbações causadas na solução candidata devem deixar a nova solução em sua vizinhança, ou seja, numa mesma região. Sendo assim, o método apenas refina a solução candidata inicialmente encontrada, sem explorar novas regiões do espaço de busca.

Uma variante do *hill climbing* padrão é o *hill climbing* iterativo. Neste caso, o algoritmo trabalha com várias soluções iniciais e armazena a melhor solução obtida. Esta alteração causa uma melhoria em termos de capacidade de busca global, uma vez que as soluções candidatas são, de início, uniformemente distribuídas ao longo do espaço de busca.

Outra variação do *hill climbing* padrão é o *hill climbing* estocástico, que faz uma escolha probabilística da solução candidata **x***'*. Assim, mesmo que a nova solução candidata seja pior que a atual solução, ela deverá assumir a posição de solução atual com probabilidade *p*. Este processo não guloso permite o escape de ótimos locais. A probabilidade *p* depende da qualidade relativa entre **x***'* e **x**, ou seja, da diferença entre os valores retornados pela função objetivo quando aplicada a **x***'* e a **x**, de acordo com a equação (2.10):

$$p = \frac{1}{1+exp\left(\frac{F(\mathbf{x})-F(\mathbf{x}')}{T}\right)} \qquad (2.10)$$

onde *F(x)* é a função objetivo, **x** a melhor solução encontrada, **x***'* a solução encontrada, e *T* um parâmetro de controle do declínio da função exponencial . O parâmetro *T* é mantido fixo durante o processo de busca, e, quanto maior for seu valor, menor será a relevância da diferença de avaliação entre **x** e **x***'*. Para *T* muito grande, a probabilidade *p* tende a 0,5, que torna o método semelhante a uma busca cega.

As alterações propostas promovem melhorias na capacidade do algoritmo alcançar novas regiões de busca, ou seja, ampliam a capacidade do algoritmo realizar a busca global. Outra técnica que também possui essa capacidade é o *simulated annealing*, que será visto na próxima seção.

2.2.2.2 Simulated Annealing

Annealing, ou recozimento, é um processo térmico que tem por objetivo obter sólidos em estados de baixa energia (Castro L. N., 2006). Para alteração da temperatura, o sólido passa por banhos térmicos quentes até que atinja a temperatura de fusão, e outros banhos térmicos são utilizados para a redução cuidadosa da temperatura até que as partículas formem por si mesmas um

arranjo, que deve ser um estado fundamental do sólido. Este estado só é obtido se a temperatura máxima for suficientemente alta e o resfriamento suficientemente lento. De outra forma, o sólido fica em um estado metaestável diferente do fundamental.

Tendo por base esse processo, Kirkpatrick (Kirkpatrick, Gelatt, & Vecchi, 1983) e Cerny (Cerny, 1985) desenvolveram de forma independente o *simulated annealing*. O método começa gerando uma solução inicial, à qual são impostas pequenas perturbações aleatórias, gerando uma nova solução. Caso esta nova solução seja melhor do que a antiga, ela passará a ser a solução atual; caso não seja ainda, haverá uma chance de ser aceita dependendo de uma distribuição de probabilidade, como a mostrada na equação (2.10).

Esta distribuição de probabilidade depende dos níveis de energia do sistema (representada pela função custo) e da temperatura de *annealing*. Quanto maior for a temperatura, maior será a chance de a nova solução ser aceita. Quanto menor for a temperatura, mais seletivo será o algoritmo, que, no limite, só aceitará novas soluções se estas forem melhores do que as antigas.

Algoritmo 2.3: *Simulated Annealing*

1) **Gerar** um ponto x em posição aleatória dentro da região factível do problema
2) **Aplicar** uma pequena perturbação no ponto x, gerando o novo ponto x', que esteja na vizinhança de x e seja factível
3) **Calcular** $\Delta F = F(x') - F(x)$
3.1) **Se** $\Delta F < 0$, então x' torna-se o ponto atual (x = x')
3.2) **Se** $\Delta F > 0$ e $\exp(-\Delta F/T) > a$, então x' torna-se o ponto atual (x = x')
4) **Reduzir** temperatura T, T=k.T

onde $F(x)$ é a função objetivo que se deseja minimizar, T é a variável temperatura, $k \in [0,1)$ é fator de variação de T e a é um número aleatório gerado entre [0,1]. O parâmetro T começa com um valor elevado e é lentamente reduzido por uma aplicação sucessiva de k durante o processo de busca.

O algoritmo termina quando um número fixo de iterações for atingido, quando não acontecer melhora significativa por várias iterações ou quando a temperatura T atingir seu valor mínimo.

O método de *simulated annealing* possui um comportamento voltado à busca local quando está em baixa temperatura, por sua estrutura probabilística. Por outro lado, o algoritmo possui

comportamento voltado à busca global quando está em temperatura elevada tendendo a aceitar pontos com pior avaliação, e, desta forma, conseguindo eventualmente ótimos locais.

A seguir outra heurística com capacidade de busca global, a busca tabu.

2.2.2.3 Busca Tabu

A busca tabu é uma metaheurística baseada em memória, que lhe permite o escape de pontos de ótimo local. A versão atual do processo foi formulada por Fred Glover em 1986 (Glover & Laguna, 2004).

O método de busca gera aleatoriamente uma solução candidata inicial e, a cada iteração, analisa a vizinhança desta solução candidata. Então, uma nova solução candidata é selecionada, e sua vizinhança também é analisada. Esse processo é repetido até que um critério de parada seja satisfeito.

No processo de análise da vizinhança, uma memória auxiliar é utilizada, o que permite que o método evite ótimos locais armazenando os pontos já visitados, permitindo que se percorram pontos piores durante o processo de busca. Desta forma, quando um novo ponto é gerado, a memória auxiliar é consultada com o objetivo de verificar se o ponto gerado já foi visitado em iterações passadas. Caso o ponto gerado esteja armazenado na memória auxiliar, este ponto já foi visitado, e, portanto, deve ser descartado, gerando–se um novo ponto.

Esta memória pode ser dividida em memória de curto e de longo prazo. A memória de curto prazo, ou lista tabu, pode ser construída na forma de uma lista circular, que armazena os últimos *pt* pontos visitados, onde *pt* é chamado de período tabu. O período pode ser fixo ou variar durante a execução do algoritmo; entretanto, se for muito pequeno, caso a probabilidade de revisitas será alta, e se for muito grande, pode deteriorar a solução por inviabilizar a volta a regiões promissoras.

Quando a lista tabu está cheia e existe a necessidade de incluir um novo elemento, o mais antigo sai. As soluções armazenadas na lista tabu estão proibidas de serem revisitadas. Com o processo de retirada da lista tabu, pontos que estão proibidos passam a ser permitidos.

A memória de longo prazo complementa as informações armazenadas na memória de curto prazo, e é utilizada para implementar tanto estratégias de intensificação como de diversificação

(Glover & Marti, 2006). As estratégias de intensificação favorecem a busca local, enquanto as de diversificação favorecem a busca global.

A memória de longo prazo baseada em frequência armazena informações sobre a ocorrência de eventos. Estes eventos podem ser a frequência de transição de uma variável ou frequência de resistência da variável. A frequência de transição registra o número de vezes que uma variável da solução candidata foi modificada; já a medida de resistência armazena a quantidade iterações em que outra variável permaneceu inalterada. Assim, é possível detectar movimentos preferenciais, levando em conta a influência do movimento na qualidade da solução.

As medidas de resistência são úteis no processo de intensificação, que visa favorecer o processo de busca em regiões historicamente boas. Já as medidas de transição fornecem informações para a visita em regiões inexploradas, ou seja, favorecem o processo de diversificação. Assim, a intensificação busca regiões conhecidamente promissoras, enquanto a diversificação regiões novas.

Outra estratégia para aumentar a diversificação é aumentar o tempo de permanência de um ponto na lista tabu, ou seja, aumentar o período tabu. Entretanto, isto pode impedir o algoritmo de revisitar regiões de boa qualidade caso os pontos desta região permaneçam na lista tabu por muito tempo. Para contornar esta situação, criou-se um mecanismo que permite a eliminação do rótulo tabu de certos pontos que satisfaçam um critério pré-estabelecido. Este mecanismo chama-se critério de aspiração, podendo ser definido e aplicado de maneiras diferentes, dependendo do objetivo do algoritmo. Um critério de aspiração muito utilizado permite um movimento tabu, ou seja, um movimento para um ponto que está na lista tabu, desde que o valor da função objetivo melhore.

Normalmente, a busca tabu para após um número especificado de iterações ou após não existir melhoria por algumas iterações. Em suma, para projetar um algoritmo de busca tabu, é necessário especificar os seguintes componentes básicos:

1) Critério de escolha da próxima solução vizinha;

2) Seleção dos atributos do movimento;

3) Memória de curto prazo para armazenar as regras de proibição (lista tabu);

4) Número de iterações que um atributo selecionado permanecerá proibido (período tabu).

5) Critério de aspiração.

O Alg. 2.4 apresenta um possível pseudocódigo.

Algoritmo 2.4: Busca Tabu

```
1) Selecionar uma solução inicial s ∈ S. Faça s*= s.
2) Gerar um subconjunto V ⊆ N(s) tal que cada elemento de V não é Tabu ou
satisfaz o critério de aspiração.
3) Escolher a melhor solução v ∈ V
4) Se v é melhor que s* então faça s* = v
5) Faça s = v
6) Atualizar a Lista Tabu.
7) Se o critério de parada for satisfeito retorne s*, caso contrário retor-
nar ao passo 2.
```

onde *S* é o espaço factível, **s** é uma solução candidata, **s*** é a melhor solução encontrada, *N(***s***)* é o conjunto com os elementos vizinhos de **s**, **V** é um subconjunto de *N(***s***) e* **v** é o melhor vizinho de **s**.

2.2.3 Métodos Imunológicos

Castro e Timmis definem os Sistemas Imunológicos Artificiais (SIAs) como sistemas adaptativos inspirados na imunologia teórica e em funções, princípios e modelos imunológicos aplicados na resolução de problemas (Castro & Timmis, 2002b).

Desta forma, um método, para ser considerado imunológico, deve ser desenvolvido através da incorporação de ideias da imunologia teórica e/ou experimental (Coelho, 2011). SIAs possuem mecanismos de manutenção de diversidade da população e as soluções localmente ótimas tendem a ser preservadas na população.

Discutimos a seguir alguns exemplos que serão relevantes para esta tese.

2.2.3.1 CLONALG

CLONALG (*CLONal selection ALGorithm*) é um algoritmo inspirado no princípio da seleção clonal do sistema imunológico humano, que pode ser aplicado a problemas de reconhecimento de padrões e otimização (Castro & Von Zuben, 2002).

O algoritmo pode utilizar duas populações, uma população de antígenos Ag, e uma população de anticorpos Ab, mas, para a aplicação em problemas de otimização, é necessário trabalhar apenas com a população de anticorpos. Este será o caso descrito nesta seção.

O método começa gerando anticorpos aleatoriamente no espaço factível do problema; assim, é criada a população inicial, que é o primeiro conjunto de soluções candidatas. Então, os n_1 melhores anticorpos são selecionados para clonagem, gerando n_2 clones por anticorpo. Os clones são em seguida hipermutados, ou seja, sofrem perturbações estocásticas: este processo realiza uma busca local ao redor anticorpo clonado.

No próximo passo, o melhor clone hipermutado é comparado com seu anticorpo gerador: caso seja melhor, ele o substituirá na população de anticorpos. Por fim, n_3 novos anticorpos gerados aleatoriamente são inseridos na população periodicamente, substituindo os n_3 piores anticorpos. O Alg. 2.5 traz uma representação computacional.

Algoritmo 2.5: CLONALG

```
1) Gerar a população de anticorpos Ab aleatoriamente dentro da região factível
2) Enquanto critério de parada não satisfeito faça
     2.1) Avaliar Ab
     2.2) Selecionar os n1 Abs melhor avaliados
     2.3) Para cada anticorpo selecionado
      2.3.1) Gerar n2 clones
      2.3.2) Hipermutar os clones
      2.3.3) Avaliar os clones hipermutados
      2.3.4) Selecionar o melhor clone
      2.3.5) Substituir o anticorpo, caso o clone seja melhor avaliado
     2.4) Fim do para
     2.5) Se critério de periodicidade atingido
     2.5.1) Gerar aleatoriamente novos n3 anticorpos
     2.5.2) Inserir os novos anticorpos na população
3) Fim do enquanto
```

Primeiramente, cria-se uma lista com os n anticorpos ordenados de acordo com os valores da função objetivo, da melhor avaliação para a pior, então faz-se o cálculo de n_2 da seguinte forma:

$$n_2 = \sum_{i=1}^{n} round(\frac{\beta n}{i}) \qquad (2.11)$$

onde β é um fator multiplicativo, n é o total de anticorpos da população, *round(.)* representa o arredondamento para o inteiro mais próximo e i é a posição do anticorpo no grupo. Quanto melhor for o anticorpo, menor será i e mais clones serão gerados.

A equação abaixo pode ser utilizada para a hipermutação em domínios reais:

$$\mathbf{c}_i(d) = \mathbf{c}_i(d) + (exp\left(-abs\left(\frac{1}{F(\mathbf{Ab}_i)}\right)\right)N(0,1) \qquad (2.12)$$

onde $\mathbf{c}_i$ é o clone i, *N(0,1)* gera um número aleatório gerado por uma distribuição normal com média zero, desvio padrão um, $F(\mathbf{Ab}_i)$ é o valor da função objetivo no anticorpo $\boldsymbol{Ab}_i$, e d é a dimensão do problema.

Em aplicações para problemas multimodais, sugere-se que seja desconsiderada a hierarquia dos anticorpos. Assim, todos os anticorpos serão selecionados para a clonagem, e todos vão gerar a mesma quantidade de clones, tomando-se $i = 1$ na equação (2.11) (Castro & Von Zuben, 2002).

O CLONALG realiza sua busca local com o processo de hipermutação dos clones, e promove a busca global com a inserção de novos anticorpos na população. Na próxima seção, será apresentado um algoritmo com a mesma inspiração, mas dotado de uma modificação importante, o potencial de controle dinâmico da população.

2.2.3.2 Opt-aiNet

O algoritmo opt-aiNet (*Artificial Immune NETwork for Continuous OPTimization*), proposto por (Castro & Timmis, 2002), foi desenvolvido para resolver problemas multimodais. Uma diferença importante do algoritmo em relação ao CLONALG é o controle dinâmico da população. Este processo favorece a manutenção da diversidade, característica relevante em processos de busca por múltiplos ótimos.

O algoritmo começa gerando uma população inicial de anticorpos no espaço de busca, sendo os anticorpos avaliados em seguida pela função objetivo. Então, ocorre o processo de clonagem: cada anticorpo deverá gerar n_2 clones, sendo este um parâmetro definido pelo usuário. O próximo passo é a hipermutação de todos os clones gerados. Após a avaliação dos clones, ocorre a substituição dos anticorpos gerados em caso de melhoria.

A seguir, é calculada a diferença da média do valor da função objetivo de todos os anticorpos da iteração passada e da iteração atual: caso esta diferença seja menor que um limiar ε definido como parâmetro, executa-se o processo de supressão, que elimina anticorpos muito próximos.

Para a supressão, é calculada a distância euclidiana entre dois anticorpos. Caso seja menor do que σ, outro parâmetro do algoritmo, o anticorpo com pior avaliação é eliminado da população. Isto acontece até que todos os anticorpos sejam comparados. Então n_3 novos anticorpos gerados aleatoriamente são incluídos na população. O Alg. 2.6 apresenta uma implementação possível.

Algoritmo 2.6: OptAinet

```
1) Gerar a população de anticorpos Ab aleatoriamente dentro da região factível
2) Enquanto critério de parada não satisfeito faça
     2.1) Avaliar Ab
     2.2) Selecionar os n1 Abs melhor avaliados
     2.3) Para cada anticorpo selecionado
      2.3.1) Gerar n2 clones
      2.3.2) Hipermutar os clones
      2.3.3) Avaliar os clones hipermutados
      2.3.4) Selecionar o melhor clone
      2.3.5) Substituir o anticorpo, caso o clone seja melhor avaliado
     2.4) Fim do para
     2.5) Se erro médio < ε
      2.5.1) Aplicar supressão
      2.5.2) Gerar aleatoriamente novos n3 anticorpos
      2.5.3) Inserir os novos anticorpos na população
     2.6) Fim do se
3) Fim do enquanto
```

No pseudocódigo, n_1 é a quantidade de anticorpos que será clonada, n_2 é a quantidade de clones gerados para cada anticorpo, n_3 é a quantidade de novos anticorpos inseridos na população, ε define a diferença mínima da avaliação média da população entre iterações.

A equação (2.13) pode ser utilizada para a hipermutação no domínio dos reais:

$$\mathbf{c}_i = \mathbf{c}_i + (\beta \,.\exp\left(-\text{norm}\big(F(\mathbf{c}_i)\big)\right) \,.N(0,I)) \qquad (2.13)$$

onde $\mathbf{c}_i$ é o clone i, *N(0,I)* gera um número aleatório segundo uma distribuição normal com média zero, desvio padrão um, e I é um vetor com *d* inteiros, todos iguais a um, *d* é dimensão do problema, n*orm(.)* normaliza *F(*$\mathbf{c}_i$*)* em [0;1]. β é um parâmetro de controle da amplitude da função exponencial inversa, e depende do problema a ser tratado.

Caso β seja muito grande em relação ao domínio do problema, isto produzirá mutações amplas, o que favorece a busca global, mas dificulta o refinamento, e, portanto, a convergência. Caso β seja muito pequeno em relação ao domínio do problema, o algoritmo pode convergir prematuramente para um mínimo local.

Este algoritmo tem características que favorecem a busca local, a busca global e a busca multimodal. A geração de clones próximos ao anticorpo favorece o refinamento, e, por conse-

guinte a busca local. Já a inserção de novos indivíduos no espaço de busca favorece a análise de novas regiões, e, portanto, favorece a busca global. O fato do algoritmo manter subgrupos independentes na população pelo processo de supressão cada subgrupo realizando sua busca local, favorece a manutenção de diversidade, e, portanto, a busca multimodal.

2.2.4 Métodos Evolutivos

Os métodos evolutivos são inspirados no processo de evolução natural, tendo por base a teoria de Charles Darwin e a genética moderna, e utilizam de maneira explícita a noção de seleção natural (Michalewicz, 1996).

A função objetivo é chamada de função *fitness*, e cada solução candidata é denominada indivíduo (segundo a abordagem Pittsburgh, que será descrita na próxima seção). Um indivíduo é expressão de um cromossomo, que é dividido em genes.

Discutimos a seguir algumas metodologias evolutivas clássicas.

2.2.4.1 Algoritmo Genético e *Niching*

Os algoritmos genéticos foram desenvolvidos por Holland, (Holland, 1975), e têm por base elementos da genética e o conceito de seleção natural. A população inicial é, normalmente, gerada aleatoriamente dentro do espaço factível. A cada geração, pares de indivíduos são aleatoriamente selecionados para o cruzamento (*crossover*), até que se produza o número esperado de descendentes. Depois, aplica-se com certa probabilidade um operador de mutação. Este processo se repete até um critério de parada ser satisfeito. Uma implementação é mostrada no Alg. 2.7.

Algoritmo 2.7: Algoritmo Genético.

```
1) Gerar a população aleatoriamente dentro da região factível
2) Avaliar o fitness de cada indivíduo
3) Enquanto o critério de parada não for satisfeito faça
      3.1) Até gerar n descendentes
            3.1.1) Selecionar dois indivíduos
            3.1.2) Aplicar crossover
            3.1.3) Avaliar o fitness dos indivíduos gerados

      3.2) Selecionar um indivíduo
      3.3) Aplicar mutação
      3.4) Avaliar o fitness do indivíduo que com mutação
4) Fim do enquanto
```

Um ponto importante a ser esclarecido é a representação de soluções do problema. Para os algoritmos genéticos, existem duas abordagens tradicionais (Bäck, Fogel, & Michalewicz, 2000):

- Michigan: a população inteira representa uma solução do problema;
- Pittsburgh: cada indivíduo da população é uma solução do problema.

O mais comum em otimização de funções é utilizar a abordagem Pittsburgh, pois a troca de material no âmbito da população favorece o processo de busca global. Deve-se ainda definir a codificação do indivíduo, que pode ser, por exemplo, binária ou real.

No caso binário, um indivíduo é um vetor de bits, e no caso real, um indivíduo é um vetor de números reais em ponto flutuante. A representação real é a mais comum para problema de otimização de funções no domínio real, pois, segundo (Michalewicz, 1996), a codificação binária possui desvantagens quando é utilizada para resolver problemas com muitas dimensões devidos aos processos de conversão de binário para real e da quantidade de bits necessária para representar um número real. Esses fatores podem fazer com que o algoritmo convirja vagarosamente.

Tanto o *crossover* quanto a mutação são ditos operadores genéticos, e existem diversas estratégias para a aplicação destes operadores. Quatro operadores de crossover para representação real estão citados abaixo (Bäck, Fogel, & Michalewicz, 2000):

- ***Crossover* Simples de um ponto**: um ponto de corte é escolhido aleatoriamente nos cromossomos pais. Então, a primeira parte do primeiro pai unificada com a segunda parte do segundo pai formam o primeiro filho. A primeira parte do segundo pai unificada com a segunda parte do primeiro pai formam o segundo filho.

- ***Crossover* Simples de *n* pontos**: semelhante ao *crossover* simples, mas com *n* pontos de corte. A combinação das várias partes deve se dar de forma a não reposicionar o gene. Por exemplo, se um gene estava na posição 5 de um dos pais, este gene deve ocupar a posição 5 do filho gerado. Isto é indispensável em problemas com variáveis que possuem domínios diferentes.
- ***Crossover* Aritmético**: Neste operador, os cromossomos descendentes são gerados a partir de uma combinação linear dos cromossomos-pai:

$$Filho_1 = \alpha Pai_1 + (1-\alpha)Pai_2 \qquad (2.14)$$

$$Filho_2 = \alpha Pai_2 + (1-\alpha)Pai_1 \qquad (2.15)$$

onde α é um número gerado aleatoriamente entre (0,1).

- ***Crossover* Heurístico**: Neste operador, o cromossomo descendente é gerado a partir da regra:

$$Filho_1 = \alpha(Pai_2 - Pai_1) + Pai_2 \qquad (2.16)$$

onde $F(Pai_2) > F(Pai_1)$ e α é um número gerado aleatoriamente entre (0,1). Caso o *crossover* heurístico produza um filho infactível gera-se outro α.

Os dois processos apresentados a seguir são instâncias de mutação sobre os reais (Bäck, Fogel, & Michalewicz, 2000):

- **Mutação Uniforme**: o operador substitui o valor do gene escolhido aleatoriamente por um número gerado aleatoriamente com uma distribuição uniforme dentro do espaço factível do problema.
- **Mutação Gaussiana**: segundo (Michalewicz & Schoenauer, 1996), o operador modifica todos os componentes de um cromossomo da seguinte forma:

$$gene' = gene + \mathbf{n}(0,\sigma) \qquad (2.17)$$

onde $\mathbf{n}(0, \sigma)$ é um vetor de variáveis aleatórias gaussianas independentes, e σ o desvio padrão.

Outro elemento importante no funcionamento do algoritmo é o método de seleção (Bäck, Fogel, & Michalewicz, 2000). Algumas possibilidades são:

- **Roleta**: a probabilidade de um indivíduo ser selecionado é dada pela razão entre seu *fitness* e o somatório de todos os *fitness* da população.
- **Rank**: os cromossomos são ordenados pelo seu valor de *fitness*, e se faz um mapeamento entre a posição do cromossomo no *ranking* e a probabilidade de ele ser escolhido.
- **Torneio**: *n* cromossomos são aleatoriamente escolhidos, com probabilidade uniforme. O cromossomo com maior *fitness* do grupo é selecionado.

Os processos de seleção influenciam diretamente a pressão seletiva do algoritmo. A pressão seletiva pode ser mensurada pela razão entre o maior *fitness* da população e o *fitness* médio. Quando a pressão seletiva é muito pequena, o algoritmo caminha pelo espaço de busca de maneira quase "cega", o que, de certa forma, favorece a busca global. Quando a pressão seletiva é muito grande, o algoritmo tende a seguir rapidamente o ponto denotado por melhor indivíduo, caracterizando uma busca local.

Segundo Lopes (Lopes, 2006), é recomendável que, nas primeiras iterações, o algoritmo mantenha a pressão seletiva em níveis mínimos, a fim de favorecer a manutenção da diversidade, e, nas últimas iterações, aumente a fim de favorecer o refinamento.

Algoritmos genéticos, não obstante, não possuem mecanismos intrínsecos de busca multimodal. Uma forma de tratar este problema é o uso de técnicas de *niching*. Uma técnica de *niching* consiste em uma estratégia para manutenção de subgrupos populacionais. A ideia é proveniente da biologia segundo a qual os subgrupos populacionais dominam regiões a fim de manter os recursos suficientes à sua manutenção. Um exemplo poderia ser um grupo de leões que domina uma região da savana, e outro exemplo seria a boa adequação simultânea de uma população de peixes num lago próximo, um ambiente muito distinto do ponto de vista de demanda de adaptação.

Uma técnica de *niching* clássica em algoritmos genéticos é o *fitness sharing*, que consiste no compartilhamento de recursos por indivíduos semelhantes (Mahfoud, 1995). Assim, quando

dois indivíduos ficam muito próximos, eles começam a compartilhar recursos, e, para representar isso, o processo de cálculo da função fitness é alterado para:

$$F_{sh,i} = \frac{F_i}{m_i} \tag{2.18}$$

onde F_i é o *fitness* do indivíduo i, calculado utilizando a função objetivo e m_i é calculado como segue:

$$m_i = \sum_{j=1}^{N} sh(d_{i,j}) \tag{2.19}$$

onde n é o número de cromossomos da população, e $sh(d_{i,j})$ é obtido da seguinte forma:

$$sh(d_{i,j}) = \begin{cases} 1 - \left(\frac{d_{i,j}}{\sigma_{sh}}\right)^{\gamma} & Se\ d_{i,j} < \sigma_{sh} \\ 0 & caso\ contrário \end{cases} \tag{2.20}$$

onde σ_{sh} é o raio do nicho, e $d_{i,j}$ é a distância entre os indivíduos i e j, e γ é uma constante capaz de regular a forma da função de compartilhamento, normalemente igual a um.

Em (Carvalho & Freitas, 2002), existe uma análise detalhada da técnica, sendo apontadas algumas dificuldades:

- Definir adequadamente o raio do nicho, pois esta definição requer um conhecimento *a priori* da distância entre os ótimos e supõe, em geral, uma distância fixa entre eles;
- A complexidade computacional do cálculo do *fitness sharing*.

Mahfoud (Mahfoud, 1995), por sua vez reporta que algoritmos genéticos com seleção proporcional e *fitness sharing* apresentaram bom desempenho em problemas multimodais, devido à capacidade de manter nichos estáveis durante a execução do algoritmo.

Outra técnica de *niching* é conhecida como *clearing*. Ela utiliza a proximidade para a criação de nichos, mas, neste caso, apenas o *fitness* do indivíduo dominante (com melhor valor da

função *fitness*) é mantido: os demais indivíduos participantes do nicho possuem *fitness* igual a zero (Petrowski, 1996). Assim, um nicho dominado se torna pouco interessante para outros indivíduos. O método possui dois parâmetros: o raio do nicho e o número de vencedores por nicho.

O método de *clearing* possui complexidade computacional menor do que a do método de *fitness sharing*, mas os problemas referentes à definição do valor do raio do nicho permanecem. Existem muitas outras técnicas de *niching*, inclusive aquelas aplicadas a outros algoritmos, como por exemplo, o *NichePSO*, o *niching* aplicado ao PSO ou ainda o *Niching* PACO, o niching aplicado ao ACO.

2.2.5 Métodos de Inteligência Coletiva

O princípio subjacente à noção de inteligência coletiva pode ser constatado na observação de comportamentos de agentes com limitada capacidade individual, que, reunidos, apresentam comportamento coletivo inteligente (White & Pagurek, 1998), ou seja, ampliam significativamente suas capacidades individuais na interação. Por exemplo, quando apenas uma formiga não é capaz de levar o alimento ao ninho, então um grupo é reunido e consegue resolver a tarefa, ou mesmo quando patos se organizam para facilitar o voo, um talvez não fosse capaz de romper imensas distâncias, mas o grupo consegue.

A inteligência coletiva apresenta as seguintes propriedades (Serapião, 2009):

- Proximidade: os agentes devem ser capazes de interagir;
- Qualidade: os agentes devem ser capazes de avaliar seus comportamentos;
- Diversidade: permite ao sistema reagir a situações inesperadas;
- Estabilidade: nem todas as variações ambientais devem afetar o comportamento de um agente;
- Adaptabilidade: capacidade de se adequar a variações ambientais.

Um dos algoritmos que utiliza a inteligência coletiva é o PSO, visto na próxima seção.

2.2.5.1 Otimização por Enxame de Partículas

O método de otimização por enxame de partículas (PSO, do inglês *Particle Swarm Optimization*) é uma técnica de inteligência coletiva. As características da inteligência coletiva aparecem no PSO quando uma partícula utiliza informações da melhor partícula do grupo (proximidade); quando uma partícula avalia sua posição (qualidade); quando as partículas estão espalhadas pelo espaço de busca (diversidade); a velocidade de movimentação da partícula utiliza sua velocidade anterior, o que será uma inércia no movimento, e, portanto, tende-se a certa estabilidade. A questão da adaptabilidade é mais nítida em problemas dinâmicos, nos quais o ótimo muda de posição, e, sendo assim, a melhor partícula deixa de ser a melhor e outra assume esta posição. Com isto, novas orientações são dadas ao grupo, e este se adapta.

Kennedy et al. (Kennedy, Eberhart , & Shi, 2001) argumentam que um algoritmo de enxame de partículas deve conter os seguintes comportamentos:

- Avaliar – as partículas devem ter a habilidade de perceber o ambiente e estimar seu próprio comportamento;
- Comparar – os indivíduos devem utilizar os outros indivíduos da população como referencial de comparação;
- Imitar – a imitação é fundamental para a organização social e importante para o processo de aquisição e manutenção das habilidades mentais.

Estes comportamentos fundamentais podem ser encontrados no PSO, que foi desenvolvido por Kennedy e Eberhart (Kennedy & Eberhart, 1995). O método emprega uma população de partículas, cada uma correspondendo a uma solução candidata para um determinado problema. Estas partículas são capazes de se mover no espaço de busca, armazenando as posições favoráveis encontradas.

No caso da otimização por enxame de partículas, a ideia fundamental consiste em espalhar certo número de partículas no domínio da função que está sendo otimizada. Então, aplica-se uma regra de movimentação das partículas utilizando a informação do grupo. Este processo é repetido até que um critério de parada seja atingido (por exemplo, um número máximo de iterações). Uma versão original do PSO pode ser implementada como no Alg. 2.8 se todas as partículas são consideradas vizinhas (ou seja, se um padrão de vizinhança global é adotado):

Algoritmo 2.8: PSO

1) **Gerar** a população de partículas aleatoriamente dentro da região factível
2) **Avaliar** a posição de cada partícula utilizando a função de otimização desejada
3) **Comparar** a avaliação de cada partícula com sua melhor avaliação anterior (o melhor valor obtido até agora), chamado ***pbest***. Se o valor corrente é melhor do que ***pbest***, será utilizado como novo ***pbest***
4) **Comparar** a avaliação corrente com o melhor valor visitado por toda a população, chamado ***gbest***. Se o valor corrente for melhor do que ***gbest***, este será utilizado como novo ***gbest***

5) **Atualizar** a velocidade (**v**) e a posição (**x**) das partículas de acordo com as equações abaixo:

$$\boldsymbol{v} \leftarrow \boldsymbol{v} + c_1\,[\boldsymbol{pbest} - \boldsymbol{x}] + c_2\,[\boldsymbol{gbest} - \boldsymbol{x}]$$
$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{v}$$

6) **Retorne** ao passo 2) até que o critério de parada for encontrado

onde c_1 e c_2 , coeficientes de aceleração, são escalares, ***gbest*** é o melhor ponto encontrado por todas as partículas, e ***pbest*** é o melhor ponto encontrado até o momento pela partícula. Cada partícula tem um único ***pbest***, e a população tem um único ***gbest***.

Segundo Bai (Bai, 2010), é possível indicar vantagens concretas e desvantagens associadas à estratégia de PSO. Algumas vantagens são: sua relativa simplicidade (em termos de estrutura e configuração de parâmetros), sua eficiência em termos de exploração – note, por exemplo, que a melhor partícula do grupo pode transmitir informações para as outras partículas - e a utilização de uma codificação direta para resolver problemas de valores reais. A principal desvantagem é a potencial perda de diversidade no caso de se adotar uma vizinhança global.

As variações mais comuns dos algoritmos de enxame de partículas se inserem na modificação da regra de movimentação das partículas e na topologia de influência das partículas. As informações mais comuns utilizadas na regra de movimentação são: melhor posição obtida, a melhor posição visitada pelo indivíduo.

O desenvolvimento natural do campo de pesquisa em inteligência coletiva levou à proposta de alternativas para o algoritmo PSO padrão, algumas das quais serão discutidas na sequência.

2.2.5.1.1 Variantes do PSO Padrão

Masehian e Sedighizadeh, em (Masehian & Sedighizadeh, 2009), analisaram 2.315 artigos sobre PSO publicados entre os anos de 1995 e 2008. Destas publicações, 536 estavam relacionadas ao método em si, e as demais 1779 diziam respeito a aplicações do PSO original e de variantes.

Das 536 publicações sobre o método, 95 algoritmos foram destacados e organizados em 22 classes distintas, que foram reagrupadas em 4 categorias genéricas: *variável*, *partícula*, *enxame* e *processo*.

As categorias genéricas foram propostas com base nas sugestões de modificação do PSO. A categoria "variável" agrupa sugestões propostas com variáveis inteiras, variáveis *fuzzy*, propostas com restrições e sem restrições de domínio e propostas com espaço discreto e binário.

A categoria "partícula" reúne propostas com sugestões de partículas capazes de atrair e repelir outras partículas. Abrange também propostas sobre a atualização síncrona ou assíncrona das partículas e sobre o cálculo da velocidade.

As propostas da classe "enxame" referem-se à topologia do exame, e ao modo pelo qual as partículas interagem com outras partículas, como por exemplo, a população da partícula interagindo em subgrupos e ou como um único grupo, trabalhando de forma cooperativa ou competitiva.

Por fim, as propostas relativas à classe "processo" referem-se às modificações quanto ao objetivo da otimização, como resolver problemas mono-objetivo ou multiobjetivo.

Na classe "variável", Shi e Eberhart (Shi & Eberhart, 1998) propuseram a utilização de um peso de inércia ω como um fator para estabelecer uma correlação entre a velocidade atual e sua atualização:

$$\boldsymbol{v} = \omega.\boldsymbol{v} + c_1\,[\boldsymbol{pbest} - \boldsymbol{x}] + c_2\,[\boldsymbol{gbest} - \boldsymbol{x}] \tag{2.21}$$

onde c_1 e c_2 são escalares aleatórios. Quanto maior for o valor de ω, maior será a tendência de busca global (*exploration*); quanto menor for o valor ω maior será a tendência de busca local (*exploitation*) (Bai, 2010). O peso de inércia na proposta original vale um, mas, nesta proposta, é

apresentado como um novo parâmetro, que, adequadamente configurado, pode colaborar para a solução do problema.

Ainda na classe "variável", Clerc (Clerc, 1999) introduziu o PSO com fator de convergência. O fator de convergência permite que as partículas oscilem em torno de regiões aleatoriamente definidas ligadas a *pbest* e *gbest*, o que pode conduzir a uma busca local e a uma convergência mais eficiente. O fator de convergência (*CF*) e a velocidade (*v*) são calculados como a seguir:

$$CF = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \tag{2.22}$$

$$\boldsymbol{v}=CF(\boldsymbol{v} + c_1\,[\boldsymbol{pbest} - \boldsymbol{x}] + c_2\,[\boldsymbol{gbest} - \boldsymbol{x}]) \tag{2.23}$$

onde c_1 e c_2 são escalares aleatórios, e $\varphi=c_1+c_2>4$. Os resultados experimentais mostram que, em comparação com o algoritmo PSO com peso de inércia, a técnica melhora significativamente a convergência do algoritmo.

Existem diversas outras variantes do PSO e propostas correlatas na literatura, dentre as quais podemos citar: *Charged PSO* proposto por (Blackwell & Bentley, 2002), que possui partículas com carga e sem carga, e as com carga se repelem; a ideia de Lu e Hou da introdução de auto-adaptação (Lu & Hou, 2004), o operador de evolução diferencial de Zhang e Xie (Zhang & Xie, 2003), o uso de filtragem Kalman em (Monson & Seppi, 2004), os elementos não lineares dinâmicos expostos em (Chen & Feng, 2005) e a utilização do conceito de seleção natural em (Tillet, Rao, Sahin, & Rao, 2005).

Ainda existem algoritmos baseados em outras populações, como por exemplo: *Glowworm Swarm Optimization,* GSO, proposto por (Krishnanand & Ghose, 2009), baseia-se no comportamento de um enxame de vagalumes no momento da formação de casais para acasalamento. Outro algoritmo é o *Density based Fish School Search,* DFSS, proposto por Madeiro, Bastos-Filho e Lima Neto (Madeiro, 2011), nele os peixes possuem a capacidade de armazenar determinadas informações, isto com o objetivo de segregar cardumes em subcardumes, fazendo com que estes explorem diferentes nichos à procura das soluções do problema.

Na próxima subseção, será descrita a heurística inspirada no eletromagnetismo, que é conceitualmente próxima à proposta original apresentada no capítulo 3.

2.2.5.2 Heurística Inspirada no Eletromagnetismo

As heurísticas inspiradas no eletromagnetismo utilizam-se dos mecanismos de atração e repulsão fundamentados por essa teoria (Birbil & Fang, 2003). Nelas, cada ponto (solução candidata) é visto como uma partícula eletricamente carregada, com valor de carga relacionado ao valor da função objetivo no ponto. A partícula é posicionada no espaço de busca, e movimenta-se na direção da força resultante a que está sujeita. O cálculo da força resultante é dado pela soma vetorial de todas as forças atuantes na partícula.

Primeiramente, o algoritmo distribui aleatoriamente *m* partículas no espaço de busca factível, e calcula o valor da função objetivo em cada partícula, armazenando a melhor solução encontrada (Birbil & Fang, 2003). Depois, calcula a força resultante em cada partícula, em consonância com a inspiração no eletromagnetismo. Por fim, movimenta a partícula com um passo aleatório na direção da força resultante, obtendo a nova solução. Havendo melhora, a nova solução substitui a anterior, e, se necessário, a melhor solução encontrada é atualizada.

A carga de cada partícula *i* é dada por q^i: o valor desta carga determina sua capacidade de atração e repulsão. Diferentemente do eletromagnetismo, as cargas não possuem sinal, e a atração ou repulsão é definida pela posição da partícula na população. As partículas melhor avaliadas atraem e as piores repelem.

A carga q^i é calculada de acordo com a expressão:

$$q^i = exp\left(-n\frac{F(\mathbf{x}^i)-F(\mathbf{x}^{best})}{\sum_{k=1}^{m}\left(F(\mathbf{x}^k)-F(\mathbf{x}^{best})\right)}\right), \forall i \tag{2.24}$$

onde *n* é a dimensão do problema, *m* é o número de partículas, $\boldsymbol{x}^{best}$ é a partícula da população com melhor avaliação pela função objetivo e *F(x)* a função objetivo.

Devido ao fato de a carga não ter sinal, o cálculo da força resultante compara as avaliações das partículas, e, então define se a partícula j atrai ou repele a partícula i. A força resultante é definida de acordo com a equação (2.25):

$$\boldsymbol{For\text{ç}a}^i = \sum_{j\neq i}^{m} \begin{pmatrix} (x^j - x^i)\frac{q^i q^j}{\left\|\mathbf{x}^j - \mathbf{x}^i\right\|^2}, & se\ F(x^j) < F(x^i) \\ (x^i - x^j)\frac{q^i q^j}{\left\|\mathbf{x}^j - \mathbf{x}^i\right\|^2}, & se\ F(x^j) \geq F(x^i) \end{pmatrix}, \forall i \tag{2.25}$$

onde *m* é o número de partículas, q^i é a carga da partícula *i*, $\mathbf{x}^i$ é a posição da partícula i no espaço de busca, e *F(***x***)* é a função objetivo.

Desta forma, os pontos melhor avaliados atraem pontos com pior avaliação; consequentemente, a melhor solução $\mathbf{x}^{best}$ atrai todos os outros pontos.

Depois da avaliação da força resultante **Força**^{i}, a *i*-ésima partícula é movimentada na direção determinada pela resultante com passo aleatório λ, gerado a partir de uma distribuição uniforme entre 0 e 1. Cada dimensão do espaço de busca está limitada superiormente por u^k e inferiormente por l^k. O movimento segue a regra:

$$\mathbf{x}_k^i = \begin{cases} \mathbf{x}_k^i + \lambda \frac{\boldsymbol{For\text{ç}a}_k^i}{|\boldsymbol{For\text{ç}a}^i|}\left(\mathbf{u}_k - \mathbf{x}_k^i\right), & Se\ \mathbf{For\text{ç}a}_k^i > 0 \\ \mathbf{x}_k^i + \lambda \frac{\boldsymbol{For\text{ç}a}_k^i}{|For\text{ç}a^i|}\left(\mathbf{x}_k^i - l_k\right), & c.c. \end{cases} \tag{2.26}$$

onde *n* é a dimensão do espaço de busca, k é a variável inteira de iteração variando entre [1,n] , $\mathbf{x}_k^i$ é a posição da partícula *i* na dimensão *k*.

Um possível critério de parada é um número máximo de iterações do algoritmo, o que é muito comum em algoritmos iterativos. Outro possível critério é a melhoria da solução encontrada, caso a melhor solução encontrada não se altere por um número fixo de passos, o algoritmo termina.

Uma semelhança entre o PSO clássico e a abordagem eletromagnética, ambos promovem a interação mutua da população de soluções candidatas. (Birbil & Fang, 2003) (Birbil, Fang, & Sheu, 2004).

No próximo capítulo será apresentado o algoritmo proposto. Além disso, será analisado seu desempenho em relação a algoritmos utilizados para otimização de funções reais.

3 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS MAGNÉTICAS

Neste capítulo, será apresentada uma das contribuições centrais do trabalho, o algoritmo MPSO (*Magnetic Particle Swarm Optimization*). Este algoritmo é populacional, iterativo, e se caracteriza por introduzir um controlador de passos com raio de repulsão, assim como o armazenamento de direções promissoras.

O capítulo começa com a justificativa do novo método, que será apresentado em detalhes na sequência. Três testes de sensibilidade serão realizados com o algoritmo, e, por fim, apresentam-se os resultados dos experimentos.

3.1 JUSTIFICATIVA DO NOVO MÉTODO

Uma busca rápida na literatura especializada em otimização revela a quantidade de métodos de inteligência de enxame e de variações existentes. Como dito no capítulo 2, apenas como variantes do PSO, (Masehian & Sedighizadeh, 2009) destacaram 95 algoritmos. No contexto desse arcabouço de soluções, a importância de um método proposto revela-se pelo seu desempenho na prática. Por outro lado, o teorema *No Free Lunch* (Wolpert & Macready, 1997) demonstra a inexistência de um melhor algoritmo num sentido geral, sendo mais correto falar em algoritmos que apresentam melhor desempenho em determinados problemas.

Entretanto, é possível discutir em termos amplos uma contribuição estrutural do método aqui proposto, a organização das relações entre busca global, local e multimodal, todas reguladas por um único parâmetro, o raio de repulsão, que define uma região de simetria esférica em que existe apenas uma partícula. Esse raio também é utilizado para limitar o passo de movimentação da partícula.

Um raio de repulsão grande favorece a busca global, pois potencializa uma grande movimentação das partículas, e também favorece a busca multimodal, pois estimula o distanciamento das partículas e a criação de nichos. Assim, o algoritmo deve começar com raio de repulsão grande, provocando grande movimentação, e deve diminuí-lo lentamente, favorecendo o refinamento das soluções encontradas. Esta solução permite o controle do comportamento do algoritmo: maior

movimentação no início, menor movimentação no final, ou seja, escape de ótimos locais no início e refinamento dos pontos encontrados no final.

Outra solução proposta foi o armazenamento das direções promissoras. Em computação, tem-se uma máxima: "o que pode ser calculado rapidamente não precisa ser armazenado, mas o que não pode ser precisa", ou seja, há um compromisso entre uso da memória e tempo de processamento. Calcular a direção do ótimo pode ser uma tarefa com complexidade significativa, mas armazenar um vetor de direção em um espaço real com *n* dimensões requer, via de regra, pouca memória. Armazenando a direção, o algoritmo caminha com maior probabilidade de acerto do próximo passo, e, com isto, aumenta sua velocidade de aproximação do ótimo local, mesmo que temporariamente.

Assim, com base nessas duas contribuições e nos resultados experimentais obtidos, justifica-se sua proposta.

3.2 ENXAME DE PARTÍCULAS MAGNÉTICAS

O MPSO (*Magnetic Particle Swarm Optimization*) é uma tentativa de reunir "boas" características apresentadas por várias heurísticas, não para ser um método com melhor desempenho em qualquer problema, o que é ingênuo do ponto de vista do teorema "*No Free Lunch*" (Wolpert & Macready, 1997), mas simplesmente para chegar a um *modus operandi* adequado para lidar com tarefas complexas de otimização multimodal.

Algumas características dos métodos de otimização levadas em consideração foram:

- Ser populacional;
- Possuir indivíduos com raio de influência;
- Ter equilíbrio entre a busca global e a busca local;
- Possuir processo de busca iterativo;
- Possuir memória.

Isoladamente ou em subgrupos, as características listadas fazem parte das estratégias pertencentes a uma ampla gama de abordagens, como inteligência de enxame, algoritmos genéticos e sistemas imunológicos artificiais. Naturalmente, os algoritmos expressam estas características de forma diferente, e as peculiaridades do método proposto serão discutidas a seguir.

3.2.1 O Algoritmo

O algoritmo MPSO tem início com o espalhamento das *m* partículas pelo espaço de busca, por intermédio de uma distribuição uniforme, seguido do cálculo do raio de repulsão para cada partícula (Prampero & Attux, 2011a). Os limites máximos e mínimos associados a este raio são definidos com base no número de partículas e no tamanho do espaço de busca. O raio mínimo é geralmente definido como a milésima parte do raio máximo, uma escolha (não obrigatória) sustentada por análises empíricas. Este processo de cálculo é útil para diminuir o número de escolhas necessárias.

O raio de repulsão tem o papel relevante de impedir que as partículas se aproximem excessivamente umas das outras. Nada acontece quando há um cruzamento exclusivamente entre regiões de repulsão, mas, quando uma partícula invade a região de repulsão da outra, a pior delas (em termos da função objetivo) é expulsa da intersecção. O tamanho de cada região de repulsão é calculado durante a execução (obedecendo aos limites acima referidos) em termos da evolução da partícula: quando uma partícula tem seu valor da função objetivo melhorado, sua região de repulsão é ampliada; caso contrário, diminui.

Inicialmente, as partículas movem-se de acordo com as equações a seguir, e três pontos são gerados por iteração:

$$\boldsymbol{Paux_{1k}} = \boldsymbol{P_k^i} + \boldsymbol{\lambda_k^i}\,\mathbf{RaioRepulsao^i} \tag{3.1}$$

$$\boldsymbol{Paux_2} = \boldsymbol{P^i} + \boldsymbol{\lambda}\,\boldsymbol{P^i} \tag{3.2}$$

$$\boldsymbol{Paux_3} = \boldsymbol{P^i} + \boldsymbol{\lambda}\,\mathbf{RaioRepulsao^i} \tag{3.3}$$

onde $k = 1,\ldots, n$ e $i=1,\ldots, m$, sendo n o número de dimensões e m o tamanho da população. $Raio\text{-}Repulsao^i$ é a região de repulsão da partícula i, $\boldsymbol{\lambda}$ é um vetor com números aleatórios pertencentes a [-1,1], e λ_k^i é o λ calculado para cada dimensão n.

As três equações são importantes, pois geram pontos em posições diferentes dentro do espaço de busca. Desta forma, espera-se que isto favoreça o algoritmo a encontrar pontos melhores em poucas iterações, reduzindo com isto as movimentações puramente estocásticas.

Enquanto a partícula ainda não tiver encontrado um ponto melhor do que o inicial, ela se movimenta para o melhor dos três pontos gerados pelas equações (3.1), (3.2) e (3.3), mesmo que este seja pior que ou equivalente ao ponto atual. Esta técnica é útil para escapar de platôs, e força a movimentação inicial da partícula pelo espaço de busca.

Quando uma solução melhor for encontrada, a partícula a armazena, assim como a direção pela qual se deslocou. No passo subsequente, o algoritmo gera o novo ponto nesta mesma linha: neste momento, o algoritmo aumenta suas chances de gerar uma sequência promissora de passos, uma vez que a direção de melhoria é conhecida, não do ótimo; o tamanho do passo também não é conhecido.

Ainda no passo subsequente, caso o ponto gerado seja pior do que a solução atual, dois pontos perpendiculares serão estudados, no caso de muitas dimensões, a reta perpendicular será gerada com base em duas dimensões escolhidas aleatoriamente. O melhor desses dois pontos pode ser adotado, desde que seja melhor do que a solução atual, e, neste caso, a linha perpendicular também será armazenada como direção atual.

Caso nenhum ponto da linha perpendicular seja melhor do que a solução atual, a partícula perde sua direção promissora, e um novo ponto é gerado aleatoriamente dentro região de repulsão. Neste caso, há uma diminuição nessa região até que o algoritmo encontre um ponto melhor do que o atual.

Como mencionado anteriormente, caso uma partícula adentre a região da outra partícula, aquela com valor mais baixo será deslocada na direção armazenada pela melhor partícula. O tamanho desta mudança de posição deve ser suficiente para expulsar a partícula da região de repulsão, eliminando o confronto.

Por outro lado, se a melhor partícula não tem direção armazenada, a pior partícula utiliza sua própria direção para movimentar-se, com um tamanho de passo adequado para deixar o confronto. Caso nenhuma das partículas do confronto possua direção armazenada, a pior partícula será deslocada na direção da melhor partícula do grupo.

Note que o algoritmo utiliza a noção de *estado*, o que evoca o conceito de autômato finito (Hopcroft, Ullman, & Motwani, 2002). Em termos simples, no estado inicial, o algoritmo faz uma busca de caráter cego, e aceita pontos piores. Quando encontra uma solução melhor, altera seu estado e se torna "guloso", movimentando-se apenas para soluções melhores. Seguem em Alg. 3.1 as ideias na forma de um pseudocódigo.

Algoritmo 3.1: MPSO

```
1) Inicializar()
2) Enquanto o critério de parada não for satisfeito faça
    2.1) Mover partículas()
    2.2) Verificar confrontos()
3) Fim do enquanto
```

O único parâmetro do algoritmo é o número de partículas, os demais foram pré-configurados no algoritmo. Além disso, o algoritmo também precisa de um critério de parada, que pode ser certo número de avaliações da função objetivo, o número de épocas do algoritmo, ou até mesmo determinado grau de variação da melhor partícula. Cada função do pseudocódigo descrito acima será detalhada a seguir.

Função: Inicializar()

Esta função distribui as partículas no espaço das soluções factíveis de acordo com uma distribuição uniforme, e calcula os limites inferiores e superiores da região de repulsão. O raio de repulsão começa com o valor do limite superior (*ubrr*), que é calculado como segue:

$$\mathbf{ubrr}^k = \frac{(\mathbf{u}^k - \mathbf{l}^k)}{2\left(\frac{m}{n}\right)} \tag{3.4}$$

sendo u^k o limite superior do espaço de busca na dimensão k, e l^k o limite inferior do espaço de busca na dimensão k, n o número de dimensões, k =1, …, n, e m o número de partículas. O limite inferior da região de repulsão (*lbrr*) é:

$$\mathbf{lbrr}^k = 0.001\ \mathbf{ubrr}^k \tag{3.5}$$

Função: Mover partículas()

```
1) Para cada partícula faça
     1.1) Se(partícula tem direção)
     1.1.1) Gere um novo ponto nesta direção
     1.1.2) Se( o novo ponto é melhor)
          1.1.2.1) Armazene este ponto
          1.1.2.2) Aumente em 10% o tamanho da região de repulsão limitada por
ubrr
     1.1.3) senão
     1.1.3.1) Gere dois pontos perpendiculares à direção antiga
     1.1.3.2) Se(o melhor deles é melhor que o ponto corrente)
          1.1.3.2.1) Armazene o melhor e sua direção
     1.1.3.3) senão
               1.1.3.3.1) Perca a direção
     1.1.3.4) fim do se
     1.1.3) fim do se
     1.2) senão
     1.2.1) Gerar três pontos utilizando as equações 3.1,3.2 e 3.3
     1.2.2) Se( o melhor ponto é melhor do que o ponto atual)
          1.2.2.1) Armazene o melhor ponto e sua direção
     1.2.3) senão
          1.2.3.1) Se( nunca teve direção)
               1.2.3.1.1) Armazene o melhor ponto gerado
          1.2.3.2) senão
               1.2.3.2.1) Reduza em 1% a região de repulsão limitado por lbrr
          1.2.3.3) Fim do se
     1.2.4) fim do se
  1.3) fim do se
2) fim do para
```

Esta função tem por objetivo movimentar a partícula pelo espaço de busca na direção de regiões com melhor avaliação, para isto verifica se a partícula armazenou informações de uma região promissora, no caso positivo, o deslocamento da partícula será nesta direção. Caso contrário, a partícula movimenta-se aleatoriamente pelo espaço, buscando uma região promissora.

Função: Verificar confrontos()

```
  1) Para cada par de partículas faça
      1.1) Se(a distância entre o par é menor do que a maior região de repul-
são das partículas do par)
      1.1.1) Se(a melhor partícula tem direção)
            1.1.1.1) Mover a pior partícula nesta direção
      1.1.2) senão
            1.1.2.1) Se(a pior partícula tem direção)
                  1.1.2.1.1) Mover esta partícula nesta direção
            1.1.2.2) senão
                  1.1.2.2.1) Mover a pior partícula na direção da melhor par-
tícula do grupo.
            1.1.2.3) fim do se
      1.1.3) fim do se
      1.2) fim do se
  2) fim do para
```

Esta função tem por objetivo manter o espalhamento das partículas no espaço de busca - para isto promove um deslocamento quando uma partícula adentra a região de repulsão de outra.

3.2.1.1 O Raio de Repulsão

O raio de repulsão é um fator importante no funcionamento do algoritmo, e, em uma análise inicial é razoável pensar que este deveria aumentar ao longo do tempo. Isto porque, após certo número de iterações, a partícula atingiu um ponto de ótimo local, e, neste caso, deve afirmar o seu domínio expulsando as partículas que porventura se aproximarem, impulsionando-as para que procurem novas regiões promissoras.

A regra de ajuste do raio de repulsão define que, quando a partícula melhora sua posição, o raio deve aumentar para favorecer a chegada da partícula ao ponto de ótimo; por outro lado, se a partícula não melhora sua posição, o raio deve diminuir. A diminuição acontece para auxiliar o refinamento, pois a partícula pode ter atingido um ponto de ótimo, e, devido à proximidade deste, estaria se movimentando ao redor dele.

Assim, dado que a partícula está próxima do ótimo, quanto menor for a região de busca, desde que esta contenha o ótimo, maior será a probabilidade de que ele seja encontrado, por isto o raio diminui.

Outro fator importante é o tratamento de conflito: quando uma partícula invade a região de repulsão da outra, a menos apta é expulsa. Neste caso, quanto maior for o raio, mais distante será colocada a partícula mais fraca, e em um processo de aproximação do ótimo, a partícula expulsa perderia o refinamento já realizado. Entretanto, caso uma "invasão" aconteça e as partículas possuam um raio pequeno, possuindo uma das partículas uma direção promissora, a partícula mais fraca será deslocada nessa direção, e esta deverá chegar mais próximo do ótimo local. Com raio muito grande, a partícula menos apta poderá ser deslocada além do ótimo local.

Mantendo a proposta do algoritmo, o tratamento de conflito funciona em modo de cooperação no final do processo, pois promove grande agitação no início e maior interação no final. Quanto maior for o raio, menor será a chance de existir cooperação entre as partículas, pois a expulsão deverá colocar a partícula mais fraca em outra região de busca: daí a "agitação".

Após o exposto, é natural pensar em dissociar o raio de repulsão do controlador de passos, mas isto acrescentaria um parâmetro ao algoritmo e comprometeria o tratamento de conflito, que, com raio grande, deverá manter a "agitação" das partículas até o final da execução do algoritmo. Esta ideia, portanto, pode não contribuir para o refinamento das soluções encontradas.

Apesar de a separação do raio de repulsão do controlador de passo poder aumentar a complexidade de configuração do algoritmo, ela pode eventualmente ser útil em determinados problemas, e, portanto, esta modificação fica como perspectiva para um novo trabalho.

3.3 ANÁLISE DE SENSIBILIDADE

Nesta seção, será analisado o comportamento do algoritmo MPSO diante da variação de alguns parâmetros. Para os experimentos iniciais, as funções foram definidas em 10 dimensões. Para o cálculo da média e do desvio padrão, foram realizadas 30 execuções do algoritmo.

3.3.1 Raio de Repulsão

Para a análise de sensibilidade com respeito ao raio de repulsão, foram utilizadas funções de referência em otimização e a função F1, que foi desenvolvida para ser um caso pouco favorá-

vel ao algoritmo, pois é unimodal e não possui mínimos locais. Todas as funções estão descritas no Apêndice A desta tese (Molga & Smutnicki, 2005) (Tang, et al., 2008).

A configuração do algoritmo precisa apenas do número de partículas, que, neste caso foi igual a 10. A execução se limita a 100.000 avaliações da função.

Tabela 3.1: Média e desvio padrão dos resultados obtidos com a variação do RR (raio de repulsão).

Funções	Algoritmo MPSO					
	Global	RR = 1	RR = 10	RR = 50	RR=calculado	
Esfera	0.00	1145.93 ± 1126	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	RR=100
Rosenbrock	0.00	6.55E07 ± 7.8E07	**0.24 ± 0.26**	0.56 ± 0.54	1.05 ± 1.50	RR=100
Griewank	0.00	141.72 ± 43.32	0.00 ± 0.01	**0.00 ± 0.00**	0.01 ± 0.03	RR=600
Schwefel	-4189.82	-1930.12 ± 369	-2945 ± 283	-2969 ± 246	**-4126±175**	RR=500
Rastrigin	0.00	37.78±11.70	**0.00 ± 0.00**	**0.00 ± 0.00**	0.00 ± 0.02	RR=5.12
F1	0.00	**0.10±0.07**	0.58 ± 0.23	1.08 ± 0.33	0.42 ± 0.19	RR=5.5
Schaffer	1.00	0.85±0.03	0.98 ± 0.01	**1.00 ± 0.00**	0.98 ± 0.01	RR=10

Para o cálculo do raio de repulsão, foi utilizada a amplitude do domínio da função, dividida pelo dobro da razão entre o número de partículas e a dimensão do problema (vide equação 3.4). Como, neste caso, o número de partículas é igual à dimensão do problema, o raio de repulsão calculado é a metade da amplitude do domínio da função.

Note que os valores 10, 50 e o calculado, para o raio de repulsão, não provocaram grande impacto geral na solução do problema. No caso da função *Schaffer*, o raio igual a 50 apresentou o melhor desempenho; para a função *Schwefel*, o melhor desempenho foi obtido com o raio igual a 500.

Entretanto, o raio de repulsão igual a 1 limitou significativamente o funcionamento do algoritmo, uma vez que, com o raio muito pequeno, diminui a probabilidade de acontecerem conflitos. Além disso, com o passo muito limitado desde o início, o algoritmo evolui pouco, apenas refina as soluções encontradas no espalhamento inicial das partículas. Assim, fica muito dependente da distribuição inicial das partículas.

O melhor desempenho na função F1 deve-se ao fato já apresentado no parágrafo anterior. Entretanto, como a função não tem mínimos locais, todos os pontos tendem vagarosamente para o máximo global. Com o raio de repulsão muito pequeno, a expulsão não chega a retirar a partícula da região de conflito, mas desloca levemente a partícula, deixando-a, mesmo após perder o conflito, em condições de competir pelo ótimo local.

Tabela 3.2: Melhor e pior caso dos resultados obtidos com a variação do RR (limite supeior do raio de repulsão).

Funções	Algoritmo MPSO									
	Global	RR = 1		RR = 10		RR = 50		RR=calculado		
		Melhor	Pior	Melhor	Pior	Melhor	Pior	Melhor	Pior	
Esfera	0.00	0.00	4372.18	0.00	0.00	0.00	0.00	0.00	0.00	RR=100
Rosenbrock	0.00	9.07	2.29E08	0.00	0.78	0.00	2.37	0.00	8.06	RR=100
Griewank	0.00	67.76	239.03	0.00	0.03	0.00	0.00	0.00	0.17	RR=600
Schwefel	-4189.82	-2727	-1079	-3737	-2490	-3459	-2521	-4189	-3479	RR=500
Rastrigin	0.00	14.16	59.35	0.00	0.00	0.00	0.00	0.00	0.15	RR=5.12
F1	0.00	0.00	0.31	0.16	1.14	0.52	1.71	0.10	1.02	RR=5.5
Schaffer	1.00	0.92	0.82	0.99	0.96	1.00	1.00	0.99	0.96	RR=10

Para os raios de repulsão com tamanho 10 ou maiores, os melhores resultados apresentados na Tab. 3.2 são, de forma geral, bons. Na maioria dos casos, o algoritmo atinge ou chega muito próximo ao ótimo global da função.

Após os experimentos, é plausível concluir que o algoritmo tem baixa sensibilidade a modificações do raio, desde que este não limite excessivamente os movimentos das partículas. Assim, caso não seja adotado o cálculo proposto para o raio de repulsão, pode-se usar, a princípio, um raio relativamente grande.

3.3.2 Número de Partículas

Este experimento analisa a sensibilidade do algoritmo em relação ao número de partículas. O cálculo do raio de repulsão utiliza o número de partículas, e, da forma como é feito, quanto maior for o número de partículas, menor será o raio de repulsão.

As funções foram escolhidas com base no desempenho do algoritmo na Tab. 3.2. O MPSO apresentou melhor desempenho com a Griewank e pior com a F1, daí a escolha.

Tabela 3.3: Média e desvio padrão dos resultados do MPSO executado com diversas configurações de partículas.

Funções	Algoritmo – MPSO					
	Número de Partículas					
	Global	3	**5**	10	15	60
Griewank	0.00	0.35±0.22	**0.00±0.01**	0.00±0.02	0.01±0.02	0.04±0.05
F1	0.00	0.84±0.16	**0.06± 0.04**	0.45±0.23	0.55±0.21	1.05±0.31

Note que, existe um equilíbrio para a quantidade de partículas. Com três partículas, o algoritmo não mostrou bons resultados, mesmo com mais avaliações da função objetivo por partícula, uma vez que, com a mesma quantidade total de 100.000 avaliações, quanto menor for o número de partículas, mais avaliações serão destinadas a cada partícula.

Por outro lado, com sessenta partículas, os resultados também não foram os melhores. Isto é um indicativo da capacidade de melhoria das soluções iniciais que o MPSO apresentou. Com mais partículas espalhadas pelo espaço de busca, maior será a chance de posicionar pelo menos uma partícula próxima ao ótimo local; apesar disto, o algoritmo conseguiu reposicionar as partículas de forma mais adequada neste experimento. Pela forma como é feito o processamento de conflitos, um número grande de partículas prejudicou o desempenho do algoritmo em função monomodal.

Assim, em relação à quantidade de partículas, o equilíbrio em relação ao tamanho do espaço e quantidade de avaliações da função objetivo deve ser utilizado para a configuração do parâmetro.

3.3.3 Número de Avaliações

Esta análise mostra o desempenho do algoritmo quanto a sua capacidade de melhoria da solução encontrada com o acréscimo de iterações. Em problemas práticos, pode ser eventualmente essencial encontrar a solução, mesmo utilizando uma quantidade maior de iterações.

Tabela 3.4: Média e desvio padrão dos resultados do MPSO com 1.000.000 de avaliações e 10 dimensões.

Funções	Algoritmo - MPSO – 1.000.000 avaliações		
	Global	5 partículas	60 partículas
Griewank	0.00	0.00±0.00	0.00±0.00
F1	0.00	0.00±0.00	0.00±0.00

Foram utilizadas 5 e 60 partículas, mantendo-se a proposta de analisar um bom resultado e um ruim. Note que, nos dois casos, como mostra a Tab. 3.4, o MPSO conseguiu atingir o ótimo global nas 30 execuções do algoritmo: por isto, o desvio padrão é igual a zero.

Com o objetivo de permitir a avaliação de um cenário mais desafiador, a Tab. 3.5 apresenta os resultados obtidos analisando as funções objetivo em 30 dimensões, lembrando que os testes anteriores desta seção foram feitos com 10 dimensões.

Tabela 3.5: Média e desvio padrão dos resultados do MPSO com 5 partículas e 30 dimensões.

Funções	Algoritmo - MPSO - 5 partículas			
	Global	Número de avaliações da função		
		1.000.000	2.000.000	3.000.000
F1	0,00	1.17±0.93	0.69± 0.23	0.57±0.09

Apesar do MPSO não atingir o ótimo global em 30 dimensões, ele não estagnou, mostrando sua capacidade de refinamento mesmo após um número significativo de iterações. Neste quesito, quanto maior for o número de avaliações, melhor será a aproximação do ótimo, desde que este já não tenha sido atingido.

3.4 EXPERIMENTO 1

Cinco algoritmos tiveram suas performances comparadas com o MPSO, a fim de manter uma base de comparação, todos os algoritmos utilizaram 30 partículas e puderam avaliar 1.000.000 de vezes as funções objetivo (Prampero & Attux, 2011a). Os algoritmos selecionados para os experimentos serão apresentados a seguir.

3.4.1 Algoritmos

Para este experimento, foram utilizados os seguintes algoritmos:

- *Particle Swarm Optimization* - PSO;
- PSO com peso de inércia;
- PSO com fator de convergência;

- *Electromagnetism-like Heuristic* - EM;
- EM com busca local;
- MPSO.

Estes algoritmos foram selecionados por apresentarem vínculos conceituais com o MPSO: no caso do PSO e de suas variações a associação é imediata; no caso do EM (*Electromagnetism-like Heuristic*), fez-se a opção por se tratar de um algoritmo que utiliza a atração e a repulsão entre partículas.

Os algoritmos PSO, PSO com peso de inércia, PSO com fator de convergência, e o algoritmo EM foram discutidos no capítulo 2, seções 2.2.5.1, 2.2.5.1.1 e 2.2.5.1.2 respectivamente. Na próxima seção, serão expostos os resultados dos experimentos realizados com estes algoritmos.

3.4.2 Resultados

Para a obtenção dos resultados, os algoritmos foram comparados de maneira equânime, ou seja, parâmetros similares foram configurados de forma igual. Além disso, vários testes preliminares foram realizados para obter os parâmetros de configuração dos algoritmos. Todos os algoritmos foram executados 30 vezes para cada caso de teste, e a média e o desvio padrão foram calculados. Os resultados foram consolidados na Tab. 3.6, ficando em negrito o melhor desempenho do caso.

Tabela 3.6: Média e desvio padrão dos melhores resultados encontrados.

Funções	Algoritmos						
	Global	PSO	PSO com peso de Inércia	PSO com fator de convergência	EM com busca local	EM	MPSO
Esfera	0,00	0.01 ± 0.01	**0.00 ± 0.00**	0.01 ± 0.02	0.21 ± 0.02	0.21 ± 0.06	**0.00 ± 0.00**
Rosenbrock	0,00	29.14 ± 0.22	28.83 ± 0.09	29.00 ± 0.07	44.58 ± 30.95	111.14 ± 213.94	**0.00 ± 0.00**
Griewank	0,00	0.02 ± 0.04	**0.00 ± 0.00**	0.03 ± 0.06	0.37 ± 0.05	0.58 ± 0.08	**0.00 ± 0.00**
Rastrigin	0,00	0.02 ± 0.04	16.69 ± 26.21	0.02 ± 0.04	35.03 ± 13.84	39.79 ± 14.78	**0.00 ± 0.00**
Salomon	0,00	0.10 ± 0.00	0.08 ± 0.02	0.09 ± 0.00	1.09 ± 0.08	1.39 ± 0.12	**0.00 ± 0.00**
Schwefel	-12569,49	-9729.18 ± 621.07	-9612.99 ± 519.12	-9698.40 ± 432.77	-8970.60 ± 1033.75	-8071.08 ± 922.09	**-12569.47 ± 0.05**
Levy	0,00	0.70 ± 2.13	0.49 ± 1.56	0.02 ± 0.05	0.23 ± 0.20	0.18 ± 0.12	**0.00 ± 0.00**
F1	0,00	72.85 ± 11.39	51.91 ± 36.12	69.39 ± 9.45	**5.92 ± 2.85**	22.45 ± 6.80	7.08 ± 2.59

Note que o MPSO alcançou o melhor resultado para sete dos oitos problemas, ficando em segundo lugar no caso da função F1, atrás apenas para o algoritmo EM com busca local. O desempenho pode ser considerado significativo, uma vez que os algoritmos têm inspiração semelhante.

Com o objetivo de verificar a capacidade de melhoria do resultado com o aumento das iterações, o número possível de avaliações foi alterado para 3.000.000 junto à função F1, que foi escolhida por conta do pior resultado apresentado pelo MPSO nesta função. Os resultados estão apresentados na Tab. 3.7.

Tabela 3.7: Média e desvio padrão do melhor resultado apresentado em cada execução – teste com 3.000.000 de avaliação da função objetivo.

Função	Global	PSO	PSO com peso de inércia	PSO com fator de convergência	EM com busca local	EM	MPSO
F1	0,00	54.92 ± 10.66	33.18 ± 12.71	57.67 ± 12.36	5.37 ± 3.62	18.92 ± 8.98	**1.48 ± 0.28**

Note que, agora, o MPSO obteve o melhor desempenho, pois manteve boa capacidade de melhoria da solução encontrada. O limite do número de avaliações da função objetivo, em muitos casos, é um fator de comparação entre algoritmos mais do que uma restrição prática de sua apli-

cação; portanto, a capacidade de continuar evoluindo a solução encontrada, após grande quantidade de iterações, credencia o algoritmo para aplicações práticas.

As tabelas deste experimento apresentam a média e o desvio padrão do melhor resultado encontrado pelo algoritmo. Desta forma, não estão explícitas nas tabelas apresentadas as soluções obtidas como um todo, ou seja, as demais soluções do ponto de vista de busca multimodal. A fim de expor o desempenho do MPSO nesse contexto de uma maneira mais ilustrativa, o algoritmo será aplicado a um problema bidimensional. Com isto, será possível analisar graficamente o desempenho do método. A função escolhida foi:

$$f(x,y) = x\sin(4\pi x) - y\sin(4\pi y + \pi) + 1 \tag{3.6}$$

onde $x,y \in [-1,1]$.

Após 2000 avaliações da função objetivo descrita na Equação (3.6), e utilizando apenas 4 partículas, o algoritmo encontrou exatamente os quatro máximos globais, como apresentado na figura abaixo. Este experimento, embora simples, ilustra o caráter de busca multimodal do algoritmo proposto.

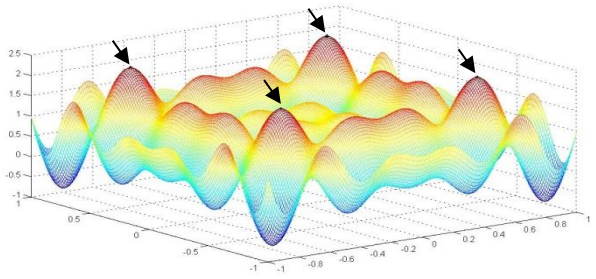


Figura 3.1: O resultado da aplicação do MPSO em um problema multimodal com duas dimensões.

Sendo assim, neste experimento, o MPSO apresentou boa capacidade de encontrar os pontos de ótimo, boa capacidade de refinamento da solução encontrada com a ampliação do número de avaliações da função objetivo e capacidade de busca multimodal. Além de encontrar os pontos de ótimo e manter a diversidade, o algoritmo conseguiu refinar de forma independente as soluções encontradas em cada nicho, ou seja, em cada região do ótimo global.

Estes resultados sugerem características relevantes e promissoras do MPSO, que serão analisadas nos próximos experimentos.

3.5 EXPERIMENTO 2

Mantendo a proposta do primeiro experimento, neste segundo experimento, é feita a comparação entre um conjunto algoritmos e o MPSO, e também uma comparação com o MPSO utilizando um operador de estimativa de distribuição (Prampero & Attux, 2011). As comparações visam encontrar pontos fortes e fracos no MPSO.

Com o objetivo de obter uma análise equânime, todos os algoritmos utilizaram 30 partículas, uma para cada dimensão, e puderam avaliar 1.000.000 (um milhão) de vezes a função objetivo. Os algoritmos utilizados serão comentados na próxima seção.

3.5.1 Algoritmos

Para este experimento foram utilizados os seguintes algoritmos:

- *Differential Evolution* - DE;
- *Estimation of Distribution Algorithm* - EDA;
- DE/EDA;
- MPSO sem a função de verificação de confrontos com ED;
- MPSO com ED

Os algoritmos foram selecionados pela popularidade e por conta do operador utilizado no MPSO, permitindo desta forma uma análise mais nítida da influência do MPSO nos resultados encontrados.

O algoritmo DE, *Differential Evolution*, foi desenvolvido por Storn e Price em 1995 (Storn & Price, 1995). O algoritmo é iniciado com a geração aleatória de pontos no espaço de busca, sendo cada ponto um indivíduo de uma população de soluções para o problema em questão (Sun, Zhang , & Tsang, 2005). O tamanho da população é um parâmetro do algoritmo.

Depois de espalhar os indivíduos sobre o espaço de busca, três indivíduos são selecionados aleatoriamente, sendo um com avaliação (valor da função objetivo no ponto) melhor ou igual à do indivíduo que está sendo analisado. Estes quatro indivíduos, o analisado e os três selecionados, serão combinados e vão gerar um novo indivíduo.

O próximo passo é a mutação, que combina o novo e o indivíduo atual de acordo com uma dada probabilidade, que também é um parâmetro do algoritmo. Finalmente, é calculado o valor da função objetivo para este novo indivíduo: caso seja melhor do que o indivíduo analisado, o novo indivíduo é incluído na população; caso contrário, o indivíduo analisado é mantido (Price K. , 1997).

O algoritmo de estimativa de distribuição, EDA, do inglês, *Estimation of Distribution Algorithm*, também é encarado como um algoritmo genético baseado na construção de modelos probabilísticos (*Probabilistic Model Building Genetic Algorithm*, PMBGA) (Pelikan, Goldberg, & Lobo, 2002). Ele se caracteriza por ser capaz de detectar padrões de evolução da população através da aplicação desses modelos. Nesse algoritmo, cada indivíduo representa uma solução candidata (Larrañaga & Lozano, 2002).

O algoritmo começa gerando a população aleatoriamente dentro do espaço de busca, sendo cada indivíduo avaliado pela função objetivo. Depois, uma amostra da população é obtida a partir da seleção de um subconjunto promissor da população. Agora um modelo probabilístico é aplicado à amostra, com o objetivo de orientar a geração de novos indivíduos. Nesta tese, a distribuição utilizada foi a normal, a média e o desvio padrão foram calculados para cada dimensão das amostras obtidas.

A fim de agregar informações obtidas pelo EDA e informações diferenciais obtidas pelo DE, o algoritmo DE / EDA foi desenvolvido (Sun, Zhang , & Tsang, 2005). De forma simples, é possível explicar o algoritmo como: a cada iteração, o algoritmo DE é aplicado com probabilidade *L*, e o EDA é aplicado com probabilidade *1-L*. Assim, o algoritmo gera novos indivíduos ora pelo algoritmo DE, ora pelo algoritmo EDA.

Por fim, o MPSO com ED consiste no MPSO já explicado na seção 3.2.1 com o EDA funcionando como uma busca local. A cada 100 ciclos, o algoritmo EDA é aplicado aos cinco melhores indivíduos da população, e a partícula gerada substitui a pior partícula da população atual.

3.5.2 Resultados

Mantendo o objetivo de realizar uma comparação equânime, os algoritmos foram configurados de forma similar, e seus parâmetros definidos por meio de ensaios preliminares. Todos os algoritmos foram executados 30 vezes para cada função, e a melhor solução encontrada em cada caso foi utilizada para o cálculo da média e do desvio padrão.

O algoritmo DE utilizou os seguintes parâmetros para todas as funções:

- Probabilidade de recombinação = 50%;
- Fator de escala F=0.6.

O EDA utilizou os seguintes parâmetros para todas as funções:

- Todos os indivíduos foram selecionados para gerar a nova população;
- O modelo de probabilidade utilizado foi baseado na distribuição normal.

Os algoritmos DE / EDA utilizaram a distribuição normal e os seguintes parâmetros para todas as funções:

- Proporção da contribuição (*Balance of contribution*) L = 50%;
- Tamanho da população = 30;
- Fator de escala F=0.6.

Tabela 3.8: Média e desvio padrão dos melhores resultados – espaço de busca com 30 dimensões.

Funções	Algoritmos					
	Global	DE	EDA	DE/EDA	MPSO com ED e sem a função de verificação de confrontos	MPSO com ED
Esfera	0,00	0.37 ± 1.26	2.86 ± 5.17	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**
Rosenbrock	0,00	308.89 ± 404.16	25.77 ± 81.36	**0.00 ± 0.00**	0.03 ± 0.02	**0.00 ± 0.00**
Griewank	0,00	0.22 ± 0.48	0.10 ± 0.12	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**
Rastrigin	0,00	98.83 ± 9.24	6.25 ± 3.08	68.64 ± 15.07	**0.00 ± 0.00**	**0.00 ± 0.00**
Salomon	0,00	0.22 ± 0.10	0.39 ± 0.21	0.11 ± 0.05	**0.06 ± 0.05**	0.09 ± 0.03
Schwefel	-12569,49	-10385.89 ± 316.91	-10708.41 ± 257.83	-8846.75 ± 923.57	-11753.31 ± 1308.24	**-12569.38 ± 0.12**
Levy	0,00	1.52 ± 0.73	0.06 ± 0.14	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**

A Tabela 3.8 mostra que os algoritmos MPSO obtiveram, para um espaço de busca com 30 dimensões, nas suas duas versões, com e sem a função de verificação de confrontos, o melhor desempenho para todos os problemas. O algoritmo DE/EDA também foi capaz de atingir o ótimo global em quatro funções. O módulo de verificação de conflito promoveu uma melhora significativa no resultado obtido para a função Schwefel.

Com o propósito de ampliar a análise estatística dos resultados obtidos, as Figs. 3.2 a 3.6 apresentam os resultados obtidos em forma de histogramas gerados com 10 execuções dos algoritmos sobre a função Rosenbrock. Em particular, estes histogramas mostram que os desempenhos de todos os métodos, exceto para as MPSO com ED sem verificação de conflitos, são comprometidos por *outliers*. Isto indica que o MPSO com ED mostrou um significativo grau de robustez, pois atingiu o ótimo global, ou chegou muito perto disto, em todas as execuções analisadas.

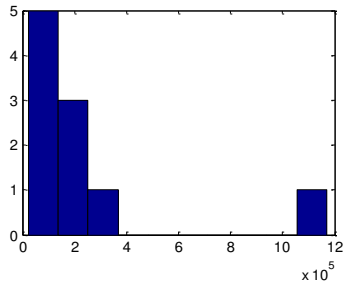


Figura 3.2: Histograma do DE

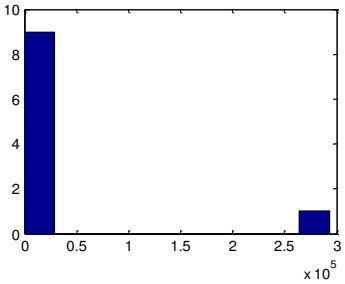


Figura 3.3: Histograma do EDA

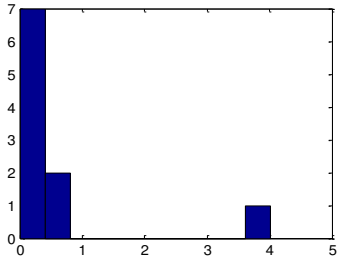


Figura 3.4: Histograma do DE/EDA

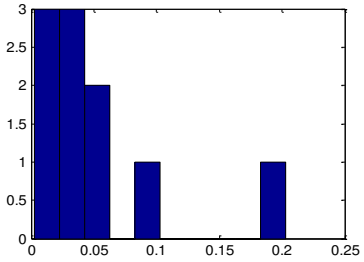


Figura 3.5: Histograma do MPSO com ED e sem a função verificar confronto

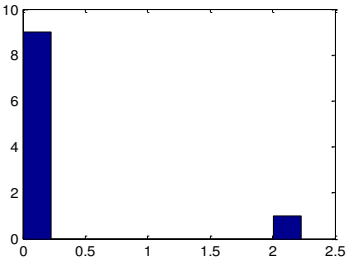


Figura 3.6: Histograma do MPSO com ED

A inclusão da função de verificação de confrontos favorece um espalhamento das partículas, o que, conforme mostra a Fig. 3.5, foi bastante útil em termos de busca multimodal.

O próximo teste foi baseado em um espaço de busca com 120 dimensões, ou seja, consideravelmente maior. O tamanho da população foi modificado para 120, seguindo ideia de manter uma partícula por dimensão, e nenhum outro parâmetro foi modificado. Os resultados estão apresentados na Tabela 3.9.

Tabela 3.9: Média e desvio padrão calculados a partir do melhor resultado encontrado em cada execução – Espaço de busca com 120 dimensões.

Funções	Algoritmos					
	Global	DE	EDA	DE/EDA	MPSO com ED sem a função de verificação de confrontos	MPSO com ED
Esfera	0,00	255.60 ± 90.80	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	0.01 ± 0.03
Rosenbrock	0,00	51897591.61± 9365980.42	160.48 ± 43.93	111.05 ± 0.17	**0.09 ± 0.08**	4.31 ± 8.34
Griewank	0,00	3.83 ± 0.91	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**
Rastrigin	0,00	1070.91 ± 38.42	924.05 ± 20.41	943.37 ± 27.16	**0.00 ± 0.00**	**0.00 ± 0.00**
Salomon	0,00	8.75 ± 0.63	0.29 ± 0.00	0.49 ± 0.00	**0.05 ± 0.04**	0.10 ± 0.00
Schwefel	-50277.948	-23363.84 ± 247.18	-8150.40 ± 337.54	-14719.93 ± 559.19	-50215.89 ± 61.45	**-50272.48 ± 17.26**
Levy	0,00	129.14 ± 15.32	**0.00 ± 0.00**	**0.00 ± 0.00**	**0.00 ± 0.00**	0.01 ± 0.01

Pelo menos uma versão do algoritmo proposto atingiu o melhor resultado em todos os problemas. A robustez do algoritmo foi indicada pelo desvio padrão, que pode ser considerado pequeno em comparação com os outros métodos. O algoritmo EDA, interessantemente, melhorou seu desempenho quando são comparadas 120 dimensões com 30 dimensões, o que parece ter sido devido ao fato de o aumento do número de indivíduos ter favorecido a estimação de probabilidade.

Finalmente, o MPSO com ED e DE/EDA foram testados com as mesmas funções em um espaço de busca com 1000 dimensões; agora, o tamanho da população foi modificado para 100 indivíduos. Os resultados, que são indicativos da capacidade de busca dos métodos em termos da média e do desvio padrão, estão apresentados na Tabela 3.10.

Tabela 3.10: Média e desvio padrão dos melhores resultados encontrados em cada execução – espaço de busca com 1000 dimensões.

Funções		Algoritmo MPSO com ED			DE/EDA		
	Global	Média ± Desvio Padrão	Pior	Melhor	Média ± Desvio Padrão	Pior	Melhor
Esfera	0,00	**0.00 ± 0.00**	0.026	0.00	1059.74 ± 60.73	1109.73	992.42
Rosenbrock	0,00	**12.04 ± 13.85**	39.68	0.42	331460.81 ± 42780.31	367249.82	284078.76
Griewank	0,00	**0.00 ± 0.00**	0.00	0.00	11.45 ± 1.10	12.72	10.73
Rastrigin	0,00	**0.00 ± 0.01**	0.04	0.00	10337.03 ± 56.98	10372.60	10271.31
Salomon	0,00	**0.12 ± 0.06**	0.30	0.10	15.55 ± 0.13	15.63	15.40
Schwefel	-418982,9	**-418958.34 ± 31.17**	-418912.52	-418982.88	-43622.91 ± 1670.93	-41694.32	-44636.23
Levy	0,00	**0.06 ± 0.06**	0.21	0.00	33.89 ± 2.67	36.30	31.00

Para problemas com 1000 dimensões, o MPSO com ED obteve um desempenho muito superior ao DE/EDA em todos os problemas, um indicativo do potencial do algoritmo resolver problemas de larga escala.

Neste experimento, o MPSO foi testado com um número significativo de funções objetivo, em espaços de busca com 30, 120 e 1000 dimensões. Seu desempenho pode ser considerado muito satisfatório em comparação os algoritmos escolhidos como *benchmark*. Além disso, o método se mostrou robusto de acordo com os histogramas apresentados.

3.6 EXPERIMENTO 3

Neste experimento, além de comparar desempenho de algoritmos da área com o MPSO, é feita uma análise de convergência do MPSO (Prampero & ATTUX, 2013). Incialmente, todos os algoritmos selecionados foram comparados com o MPSO; mantendo-se o padrão dos experimentos anteriores, todos os algoritmos utilizaram 30 partículas (indivíduos), um por dimensão, e puderam avaliar 1.000.000 de vezes a função objetivo.

Configurações similares foram feitas em algoritmos similares, e os parâmetros mais específicos foram estabelecidos por testes preliminares de desempenho.

3.6.1 Algoritmos

Para este experimento, foram utilizados os seguintes algoritmos:

- PSO;
- EM;
- DE/ EDA;
- MPSO;
- MPSO com ED;

Os algoritmos utilizados foram apresentados nas seções 2.2.5.1, 2.2.5.1.2, 3.5.1, 3.2.1 e 3.5.1, respectivamente.

3.6.2 Resultados

Os resultados aqui mostrados são provenientes de 30 execuções de cada algoritmo para cada problema, e a média e o desvio padrão foram calculados utilizando o melhor resultado encontrado em cada execução.

O PSO utilizou c*1*=0.5 e c*2*=0.2 para todas as funções. O DE / EDA utilizou uma distribuição normal e *"balance of contribution"* L = 50%, tamanho da população igual a 30 e fator de escala F=0.6 para todas as funções. O MPSO com ED executou o passo ED a cada 100 ciclos, sendo a média e o desvio padrão para o ED calculados com as melhores cinco partículas, e a nova partícula gerada substituía a pior partícula da população. Os resultados estão resumidos na Tabela 3.11, e o melhor desempenho foi colocado em negrito.

Tabela 3.11: Média e desvio padrão do melhor resultado obtido de cada execução - Espaço de busca com 30 dimensões.

Funções	Algoritmos					
	Global	PSO	EM	DE/EDA	MPSO	MPSO com ED
Esfera	0,00	0.02± 0.03	0.26±0.05	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**
Rosenbrock	0,00	28.98± 0.05	83.02±88.95	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**
Griewank	0,00	0.05± 0.08	0.55±0.09	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**
Rastrigin	0,00	0.01± 0.02	45.77±17.59	73.60±8.82	**0.00±0.00**	**0.00±0.00**
Salomon	0,00	0.10± 0.00	1.38±0.08	0.12+0.04	**0.00±0,00**	0.09±0.02
Schwefel	-12569,49	-9636.83± 709.21	-8606.34± 705.50	-10762.50 ±885.11	**-12569.42 ±0.09**	**-12569.42 ±0.09**
Levy	0,00	1.44± 3.12	0.32±0.31	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**
F1	0,00	311.70± 146.23	32.26±16.79	**0.00±0.00**	8.10±1.82	5.91±1.42

As versões do MPSO alcançaram o melhor resultado para sete dos oito problemas, ficando atrás apenas do algoritmo ED/EDA no problema da função F1. Pelos resultados apresentados na tabela acima, tem-se a possibilidade de o passo ED favorecer a convergência em problemas monomodais.

As Figs. 3.7 e 3.8 ilustram a convergência do algoritmo MPSO, informando a movimentação da melhor partícula e do valor médio de avaliação da população. Em todos os casos, o comportamento foi obtido a partir da execução do MPSO padrão.

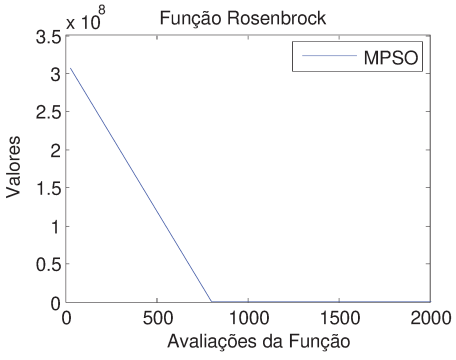


Figura 3.7: Evolução da melhor partícula.

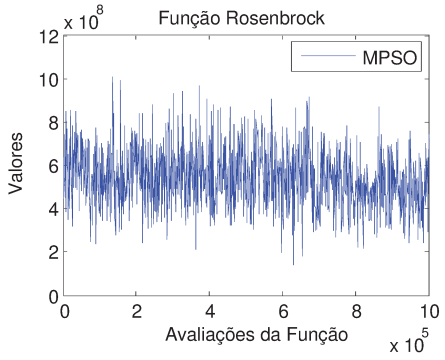


Figura 3.8: Evolução da média da população.

A melhor partícula converge rapidamente, antes de 1000 avaliações, mas a população ainda mantém a diversidade, com picos de melhora e piora. Estes picos podem ser explicados pelos confrontos entre partículas, que promovem seu espalhamento reduzindo a média. Note ain-

da que, mesmo com subidas e descidas, a média vai melhorando lentamente. Isto acontece porque o raio tende a ficar menor, o que reduz a frequência dos confrontos, fazendo com que as partículas tendam a se estabilizar no final da execução do algoritmo.

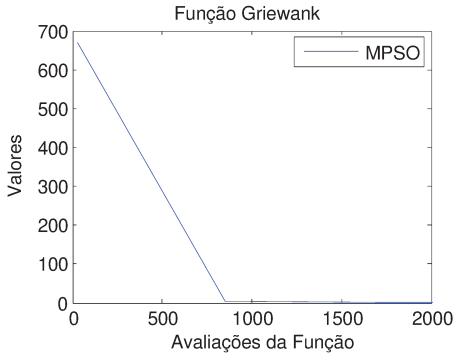


Figura 3.9: Evolução da melhor partícula.

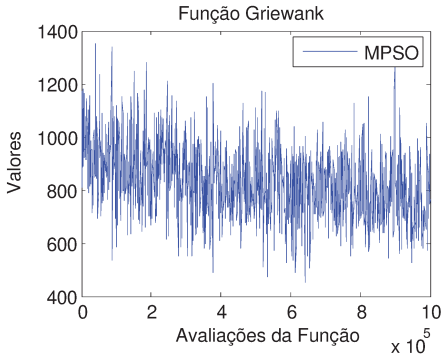


Figura 3.10: Evolução da média da população.

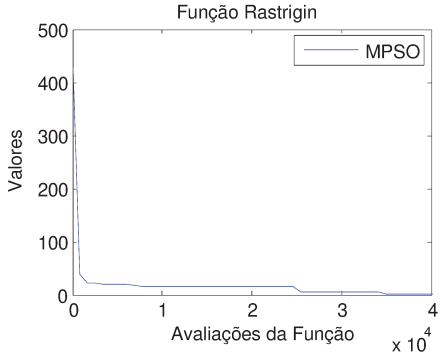


Figura 3.11: Evolução da melhor particular.

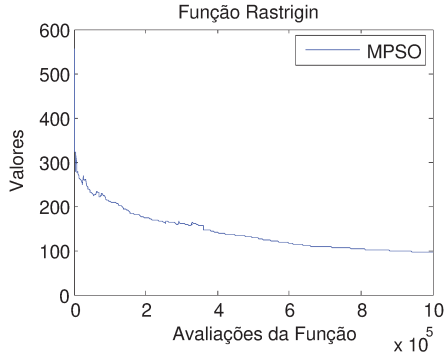


Figura 3.12: Evolução da média da população.

O caso da função *Griewank*, Figs. 3.9 e 3.10, é muito semelhante ao caso da função *Rosenbrock*. No caso da função *Rastrigin*, Figs. 3.11 e 3.12, a melhor partícula convergiu rapidamente - note os pequenos saltos durante a convergência inicial, muito provavelmente substituições da melhor partícula. Neste salto, não foi a partícula que melhorou, mas outra partícula que assumiu o posto de melhor partícula. No caso da média, houve uma melhora suave, o que denota o refinamento das soluções encontradas. No caso da função *Schwefel*, Figs. 3.13 e 3.14, a convergência da melhor partícula ocorreu até a avaliação número 1000 da função objetivo, e a média estabilizou por volta dos -8000.

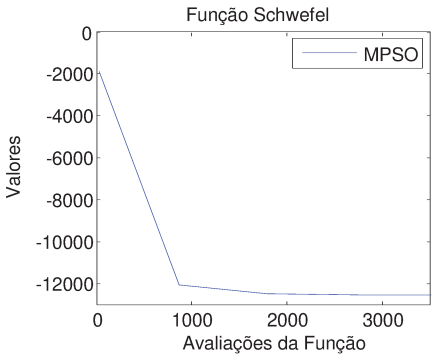


Figura 3.13: Evolução da melhor partícula.

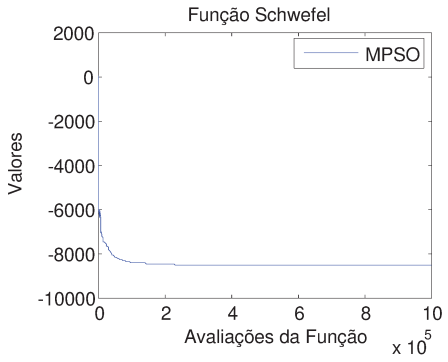


Figura 3.14: Evolução da média da população.

De forma geral, as informações contidas nas figuras apresentadas mostram uma convergência rápida para a melhor partícula, e uma convergência mais lenta para a média da população. Mostram também que a melhor partícula só evoluiu, e a população média apresentou oscilações, fato importante para a manutenção da diversidade populacional e da capacidade de escapar de ótimos locais. Isto confirma que as motivações subjacentes à construção do método foram, em alguma medida, justificadas.

3.7 EXPERIMENTO 4

Este experimento foi realizado com dados reais. O algoritmo MPSO foi utilizado para otimizar a produção de biogás do aterro Bandeirantes (Costa Junior, Prampero, Assad, & Attux, 2011).

O problema consiste na otimização da captura do metano no aterro Bandeirantes, que foi projetado para processar cerca de 122.640.000 m^3 de biogás por ano. Sendo assim, caso a produção vá além desta capacidade, o gás excedente será queimado e não aproveitado para geração de energia. Desta forma, idealmente, a produção de biogás precisaria ser tão próxima quanto possível dos 122.640.000 m^3 por ano. Para isto, um modelo simples baseado em punições foi criado, como mostrado a seguir:

```
1)     Se(Produção(ano)>122640000)
1.1)   Total(ano)=122640000 – (Produção(ano) – 122640000)/2;
2) Senão (Produção(ano)<120000000)
       2.1)Total(ano)=-100000000;
3) Fim do se
```

onde

$$F(ano) = \sum_{i=1}^{ano} total(i) \tag{3.7}$$

é a função que se deseja otimizar, e

$$Produção(ano) = L_o K \sum_{i=1}^{depósito\_ano} M_i e^{-(K(ano-i))} \tag{3.8}$$

onde *Produção(ano)* é um modelo de geração de biogás, produzido por (US EPA, 2013), *ano* são os anos de funcionamento do aterro, *depósito_ano* é o número de anos de depósito dos resíduos, e *Mi* é a massa de resíduos depositados no aterro no ano, em toneladas. L_0 é uma constante igual a 125 m^3/t de metano – de acordo com os dados do aterro Bandeirantes. (Ensinas, 2003). *K* é outra constante, igual a 0,1 ano^{-1} que denota a taxa de decaimento de acordo com os dados do aterro Bandeirantes (Ensinas, 2003).

O modelo penaliza dois cenários, o primeiro quando a produção de metano for maior que 122.640.000 m^3 de gás, e o segundo quando a produção for menor que 120.000.000 m^3 de gás. Entretanto, a segunda penalidade é proibitiva, pois a produção associada é forçada a -100.000.000. Esta é uma forma de evitar que a produção calculada pelo algoritmo de otimização seja menor que 120.000.000 m^3 de gás por ano, fazendo com que o aterro opere em lucro.

Nos experimentos realizados, o algoritmo utilizou 50 partículas e avaliou 1.000.000 de vezes a *F(ano)*. Dado o número de anos de depósito, o algoritmo calcula os depósitos anuais (Mi) para que o aterro tenha produção ótima. Os resultados obtidos nos experimentos são apresentados na tabela abaixo.

Tabela 3.12: Depósito original de resíduos no aterro Bandeirante e otimização com 12, 15 e 20 anos.

Anos de depósito	Massa em tonelada depositada por ano									
	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
Bandeirantes	1095000	1149750	1207238	1267599	1330979	1397528	1467405	1540775	1617814	1698704
12	9812246	933273	932996	933825	933715	933739	952277	916777	933829	933841
15	9811192	934445	933379	933736	933538	933655	933743	933418	935864	932303
20	9811488	937372	930010	1006057	867838	1057332	821780	932012	937239	932636

Tabela 3.13: Depósito original de resíduos no aterro Bandeirante e otimização com 12, 15 e 20 anos.

Anos de depósito	Massa em tonelada depositadas por ano									
	M11	**M12**	**M13**	**M14**	**M15**	**M16**	**M17**	**M18**	**M19**	**M20**
Bandeirantes	1783640	1872822	1966463	2064786	2168025					
12	933716	4478293								
15	932945	933765	934465	934115	1677965					
20	932617	934306	1329593	583104	926700	36557	19877	385240	246768	0

A soma dos resíduos depositados é de 23.628.527 toneladas para todos os casos, e o aterro Bandeirantes depositou seus resíduos por 15 anos, até o seu encerramento. Nas linhas com 12, 15 e 20 a quantidade de depósito foi calculada pelo algoritmo. No caso de 20 anos, o algoritmo não depositou nada no último ano, os resíduos não foram suficientes para isto neste caso.

A seguir, são mostrados os gráficos de depósitos por anos e a produção de biogás.

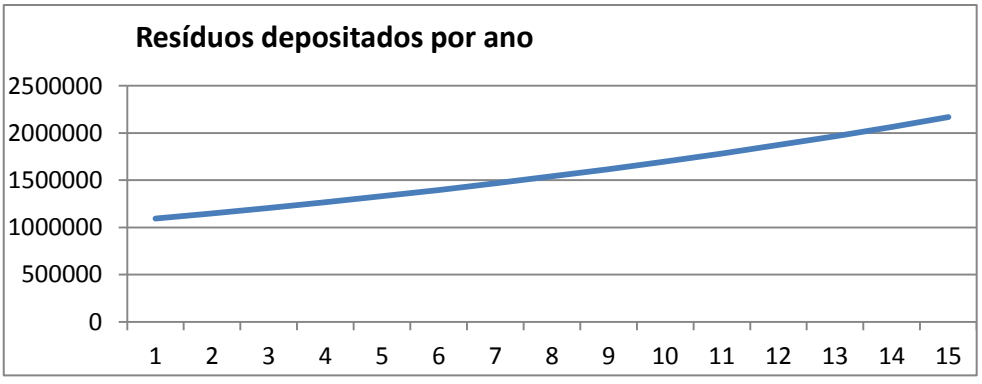


Figura 3.15: Resíduos depositados por ano no aterro Bandeirantes.

O aterro Bandeirantes foi aumentando a quantidade de depósitos de resíduos ao longo dos anos. Esta não é uma boa estratégia, pois segundo o modelo apresentados na Equação (3.8), os resíduos depositados demoram algum tempo para produzir biogás. Assim, o segundo ano deverá ter pouca produção.

Figura 3.16: Produção de biogás por ano no aterro Bandeirantes.

Os resíduos depositados no aterro Bandeirantes foram aumentando pouco a pouco, e a produção de biogás alcançou a capacidade de captura no 11° ano, não tendo sido possível manter essa produção além do 19° ano, conforme mostra a Fig. 3.16. A produção atingiu um pico, alcançando a produção de 167353975 m^3 no 15° ano, mas o aterro Bandeirantes não pode aproveitar mais do que 122640000 m^3 para geração de energia, sendo o excedente queimado. Neste caso, a produção fica acima desta capacidade por 8 anos, e abaixo nos demais. Assim, o aterro tem apenas dois momentos de produção ideal; na maioria dos casos ou está desperdiçando biogás, ou esta sendo subutilizado.

As Figuras 3.15 e 3.16 mostraram o que aconteceu no caso real do aterro Bandeirante e as próximas figuras mostram as simulações obtidas pela aplicação do MPSO no modelo matemático.

Figura 3.17: Resíduos depositados por ano na otimização com 12 anos de depósito

Figura 3.18: Metano produzido com 12 anos de depósito.

É possível notar que a otimização tentou manter a produção de biogás muito próxima da capacidade de geração do aterro Bandeirantes, e, para alcançar a produção máxima do aterro, o sistema indicou a necessidade de uma grande quantidade de depósito de resíduos no início de funcionamento. A Figura 3.17 mostra um excedente de resíduos depositados no 12° ano. Podemos observar neste estudo que o sistema manteve a produção de biogás acima da capacidade de produção de energia por 15 anos, e o pico de produção mostra que os depósitos deveriam ocorre de forma mais espalhada, ou seja, por mais tempo, por isto a próxima simulação com 15 anos de depósitos.

Figura 3.19: Resíduos depositados por ano na otimização com 15 anos de depósito.

Figura 3.20: Metano produzido com 15 anos de depósito.

O caso de depósito em 15 anos é similar ao caso de 12 anos, mas, em 15 anos, a sobra de resíduos é menor e o pico de produção também, como mostram as Figs. 3.19 e 3.20. Neste caso, assim como no anterior, o sistema manteve a produção de metano acima da captura desejável por 15 anos. Por fim, a simulação com 20 anos é mostrada nas Figuras 3.21 e 3.22.

Figura 3.21: Resíduos depositados por ano na otimização com 20 anos de depósito.

Figura 3.22: Metano produzido com 20 anos de depósito.

O estudo para 20 anos de depósito de resíduos mostra que os depósitos não foram suficientes para manter a produção acima de 12.640.000 m^3 por 20 anos. No entanto, os resultados ainda mostram que o sistema manteve a produção de metano acima da capacidade de geração de energia por 15 anos.

As três simulações utilizando o algoritmo proposto mostraram que a quantidade de resíduos é suficiente para manter a produção de metano por 15 anos, em todos os casos, muito superior ao caso real em que a produção foi superior por apenas 8 anos.

Os experimentos mostraram ainda que é possível controlar a produção de biogás através da gestão dos depósitos de resíduos do aterro, tendo por base um modelo matemático. Quando

um aterro for projetado, será possível calcular a capacidade de resíduos e a produção de biogás e a quantidade de resíduos por ano necessária para otimizar a geração de energia.

Com estas informações, será possível prever o início da geração de energia, quanto tempo terá de vida útil e quando será o retorno do investimento do aterro. A precisão dessas informações depende da qualidade matemática dos modelos usados na previsão de geração de biogás e do projeto do aterro.

Por fim, os resultados obtidos demonstram a possibilidade de otimização da geração de energia dos aterros, desde que seus projetos sejam concebidos para este propósito e em conjunto com uma melhor gestão dos resíduos.

Conforme apresentado nas três simulações, o algoritmo sugeriu uma grande concentração de resíduos nos primeiros anos e redução nos demais anos. Como a produção de resíduos é proporcional ao crescimento demográfico da região atendida pelo aterro, será difícil a aplicação direta dos resultados obtidos em um aterro, mas é possível a utilização dos resultados para melhorar o que já existe.

3.8 CONCLUSÃO

Neste capítulo, o algoritmo MPSO foi apresentado juntamente com as bases subjacentes a sua construção. De acordo com a seção 3.2.1, o algoritmo apresenta a seguintes características estruturais:

- Iterativo;
- Populacional;
- Possui memória;
- Possui controlador de passos;
- Armazena direção promissora;
- Define região de atuação da cada partícula.

Pode-se dizer que o algoritmo possui dois pontos inovadores: o armazenamento da direção promissora e o controlador de passos. Nos experimentos realizados, o algoritmo apresentou as seguintes características funcionais:

- Rápida convergência da melhor partícula;
- Boa manutenção da diversidade;
- Boa capacidade de refinamento, mesmo após a execução de muitos ciclos;
- Capacidade de solucionar problemas de larga escala;
- Simplicidade de configuração.

Tendo por base as características estruturais e funcionais, junto com o bom desempenho apresentado pelo MPSO nos problemas testados, o algoritmo mostrou-se uma ferramenta competitiva e promissora.

4 OTIMIZAÇÃO POR CLUSTERIZAÇÃO

Neste capítulo, será apresentada a segunda contribuição central a esta tese, uma nova abordagem para o problema de otimização multimodal baseada em uma formulação da tarefa de busca como um processo de agrupamento de dados (clusterização ou *clustering*). Uma vez que o processo de clusterização se liga, intrinsecamente, à noção de buscar, de maneira concomitante, múltiplos protótipos que se posicionem em regiões de maior densidade de dados, é possível explorá-lo a fim de auxiliar um algoritmo a realizar uma melhor movimentação dos pontos no espaço de busca associado a uma função objetivo.

Para isto, propomos conceber a função custo como uma função de densidade de probabilidade de dados fictícios, de maneira que o algoritmo de clusterização escolhido seria, idealmente, capaz de encontrar as regiões de maior concentração. Elas corresponderiam aos múltiplos picos da referida função de densidade, os quais, por sua vez, estão associados aos pontos modais da função objetivo original. Com isto, abre-se a possibilidade de aplicar à tarefa de otimização multimodal um vasto repertório de métodos de clusterização, especialmente como auxiliares de heurísticas de busca.

A região promissora é um indicativo da presença de ótimos locais, e, portanto, tal região precisa ser avaliada com mais detalhes. Naturalmente, este indicativo é passível de falhas, principalmente quando os ótimos locais são extremamente íngremes, e em conjunto com um espalhamento de poucos pontos pela região, uma região que deveria ser classificada como promissora pode ser descartada.

A metodologia desenvolvida utiliza um processo de clusterização modificado, que translada o centro de massa dos grupos populacionais para regiões promissoras. Este processo será descrito em detalhes nas próximas seções, bem como será testado e analisado nos experimentos realizados.

4.1 ANALOGIA ENTRE OTIMIZAÇÃO E CLUSTERIZAÇÃO

Clusterização pode ser definida em termos simples como uma tarefa de identificar agrupamentos em um conjunto de dados que, em geral, não estão rotulados (Jain, Murty, & Flynn,

1999). A formação de grupos ou classes de dados é feita tendo como base um critério de similaridade, o qual, usualmente, é definido em função de informações espaciais dos dados e envolve o posicionamento de pontos denominados protótipos. Esses pontos passam a representar os dados que, segundo o critério de similaridade, possuem maior afinidade com eles.

Embora o processo de adaptação da posição dos protótipos dependa da técnica de clusterização empregada, ele é essencialmente influenciado pela densidade de amostras em cada região do espaço de dados. Regiões onde há maior concentração de dados exercem maior força de atração sobre os protótipos mais próximos. Por outro lado, se o número de amostras em uma região for pequeno, apenas em alguns momentos o protótipo mais próximo será atraído em sua direção. Além disso, a área de cobertura de cada protótipo é inversamente proporcional à densidade de amostras: um protótipo cobre uma área maior do espaço dos dados nas regiões em que há uma menor concentração de amostras.

Por sua vez, o processo de otimização consiste em encontrar pontos com a maior ou menor avaliação de acordo com uma função objetivo (Camponogara, 2006). No caso de um cenário multimodal, o desafio é encontrar de maneira simultânea um conjunto de pontos que sejam localmente ótimos.

O processo de busca pela solução ótima é diretamente influenciado por informações extraídas da própria função objetivo. Classicamente, os algoritmos de otimização não-linear utilizam informações de primeira e segunda ordem associadas à função objetivo, tais como o vetor gradiente e a matriz hessiana (Luenberger & Ye, 2008), como visto no capítulo 2. Em contrapartida, metaheurísticas bio-inspiradas fazem uso apenas do valor da função objetivo em um dado ponto, o que, embora não ofereça o mesmo nível de conhecimento a respeito das características da função, elimina o custoso processo de estimação de derivadas (Castro L. N., 2006).

Com estas definições em mente, é possível formular o processo de otimização como uma tarefa de clusterização. Lembrando que a densidade de amostras controla o posicionamento dos protótipos, tal processo pode ser útil em otimização caso consigamos estabelecer uma conexão direta entre as regiões de maior densidade de um conjunto de dados e os ótimos locais da função objetivo.

Uma maneira de realizar esta conexão consiste em interpretar a função objetivo como uma função densidade de probabilidade fictícia cujas amostras constituem os dados para clusterização. Assim, com o propósito de obter, ao final do processo de clusterização, um repertório de protóti-

pos ocupando as melhores regiões do espaço de busca em termos do valor da função objetivo, o conjunto de dados deve ser tal que um número significativo de amostras reside nas regiões mais promissoras desse espaço, enquanto poucas amostras são tomadas nas demais regiões.

Na prática, porém, não se sabe a priori quais regiões do espaço de busca são ou não promissoras para realizar a amostragem segundo a densidade desejada. Uma maneira de contornar esta dificuldade é realizar a amostragem da função custo de maneira cega (i.e., segundo uma distribuição uniforme) e emular a densidade de pontos – que, à luz da conexão estabelecida anteriormente, reflete o quão promissora é uma dada região do espaço de busca – através da inserção de um fator de ponderação (peso) que influencie a dinâmica de ajuste de cada protótipo e que traga a informação a respeito do valor da função objetivo associado àquela amostra. Por exemplo, caso a técnica de clusterização seja baseada em um procedimento iterativo, podemos ponderar o passo de adaptação com um peso proporcional ao valor da função objetivo para simular o efeito de uma concentração maior de pontos nesta região de um conjunto de dados fictício.

A característica de atração (entre protótipo e os dados que ele representa) e repulsão (entre protótipo e demais dados) faz com que o processo de clusterização seja uma ferramenta pertinente em termos de busca multimodal. Contudo, este processo não visa identificar de forma precisa os pontos máximos da função densidade de probabilidade associada aos dados, o que, entretanto, é crucial em uma tarefa de otimização. Por isso, é pertinente combinar este estágio de otimização baseada em clusterização com uma estratégia de refinamento das soluções para alcançar estimativas mais precisas das posições reais dos ótimos.

Além disso, diferentes estratégias de clusterização podem ter características que se mostrem mais interessantes para diferentes instâncias do problema de otimização multimodal. Estabelece-se, portanto, uma sinergia entre tarefas clássicas de aprendizado de máquina que, até o presente momento, não possuem uma inter-relação conceitual tão ampla quanto seria desejável.

Exposta a proposta, segue a discussão dos métodos de clusterização que serão utilizados neste trabalho para avaliação de desempenho.

4.2 CLUSTERIZAÇÃO

O problema de clusterização pode ser considerado muito relevante nos âmbitos da aprendizagem não-supervisionada e análise estatística de dados. Basicamente, clusterização é a classificação dos dados não rotulados em grupos, chamados de *clusters*, com base em algum critério de similaridade, como, a posição no espaço (Duda, Hart, & Stork, 2001). Esta tarefa de identificar classes semelhantes em uma coleção de dados é útil em uma ampla gama de contextos, como mineração de dados, recuperação de informações, segmentação e classificação de padrões (Jain, Murty, & Flynn, 1999).

Um dos algoritmos mais simples para resolver o problema de clusterização é o *k-means*, que será apresentado na próxima seção.

4.2.1 k-means

O *k-means* é um algoritmo muito conhecido de clusterização. Em sua versão estocástica, inicialmente, este algoritmo gera *k* centros ou centroides, aleatoriamente, no mesmo espaço em que estão os dados (Duda, Hart, & Stork, 2001). Então, cada ponto do conjunto de dados é apresentado ao algoritmo, que o atribui ao domínio do centroide mais próximo. Neste momento, a posição do centroide é modificada para que este se aproxime ainda mais do ponto apresentado. Com o passar das épocas de treinamento, os centroides vão se movimentando pelo espaço e se posicionando nos "centros de massa" dos grupos de pontos que representam. Quando a condição de convergência é satisfeita, tem-se que cada centroide representa uma classe no espaço de pontos.

Este algoritmo corresponde a uma estratégia simples de agrupamento, e possui algumas dificuldades. Uma delas é que existe uma dependência significativa entre as classes obtidas e as posições iniciais dos centroides. Outra é que o número de centroides é fornecido como parâmetro do algoritmo, embora nem sempre o número de classes seja conhecido previamente. Havendo divergência entre os números de centroides e classes, poderá haver unificação ou particionamento de classes, o que é indesejável.

Não obstante essas dificuldades, o *k-means* é uma ferramenta popular por apresentar um balanço interessante entre simplicidade e desempenho, e servirá de base para a exposição da ideia de otimização baseada em clusterização. O algoritmo 4.1 apresenta um pseudocódigo de uma possível implementação.

Algoritmo 4.1: K-means

```
1) Inicialize os centroides aleatoriamente pelo espaço de busca
2) Enquanto o critério de parada não for satisfeito faça
  2.1) Associe cada ponto ao centroide mais próximo
  2.2) Recalcule a posição do centroide
3) Fim do enquanto.
```

Os centroides podem ser atualizados de acordo com a equação abaixo:

$$\mathbf{c_i}(n+1) = \mathbf{c_i}(n) + \mu(\mathbf{x} - \mathbf{c_i}(n)) \qquad (4.1)$$

onde μ, muitas vezes chamado de taxa de aprendizagem, define a amplitude do movimento em direção ao ponto **x**. Note-se que a ideia de ajustar o centroide para ser mais parecido com o padrão estabelece uma conexão com a estrutura de aprendizado competitivo que caracteriza os chamados mapas (Kohonen, 1995).

Para que o *k-means* possa ser adequadamente aplicado, alguns aspectos importantes devem ser levados em consideração. O primeiro é a representação de um determinado ponto de referência ou padrão: de um modo geral, é constituído por um vetor $\mathbf{x} = (x_1, ..., x_D)$. Segundo, a medida de similaridade, o que está fortemente relacionada com a representação do padrão. Por fim, o número *k* de centros, que também fixa o número de grupos ou classes, nos quais os padrões serão agrupados.

O algoritmo *k-means* é muito popular, por ser relativamente fácil de implementar e aplicável mesmo a grandes conjuntos de dados. No entanto, como mencionado, o método tem uma dificuldade intrínseca: a sua convergência depende fortemente da condição inicial, isto é, diferentes conjuntos iniciais de centros podem conduzir a resultados diferentes; além disso, dependendo do critério de parada, o algoritmo pode até mesmo não convergir. Outro aspecto crítico é que uma escolha inadequada do número de grupos *k* pode levar a partições inadequadas.

Dada a posição inicial dos centros, o algoritmo *k-means* tende a espalhá-los ao longo das regiões do espaço de entrada ocupado pelas amostras. Assim, a configuração final do conjunto de centroides depende da concentração de dados ao longo do espaço. Na verdade, é possível observar que, quanto mais populosa for certa porção do espaço de entrada, maior se tornará a sua força de atração, de forma que o centroide mais próximo será constantemente movido na sua direção.

Por outro lado, se o número de amostras numa região é pequeno, apenas em alguns casos o centroide mais próximo irá avançar para esta região. Outra observação importante é que os centroides associados a regiões populosas representam uma área menor, enquanto os que estão associados a regiões com pequena concentração de dados tem um "raio de ação" maior.

4.2.1.1 Otimização Baseada no *K-means* Modificado

Considere obter um conjunto de amostras de $\mathbf{S}_i$, geradas com o espalhamento de pontos no espaço de busca e posterior avaliação destes pontos pela função $F(\mathbf{x})$. Assim, os valores de $F(\mathbf{S}_i)$, i = 1, ..., N estão disponíveis. Desta forma, têm-se o ponto e sua avaliação, dados suficientes para utilizar o algoritmo de *k-means* nestas amostras.

A aplicação direta do *k-means* considerando as amostras do espaço de busca provavelmente irá alocar os centroides em regiões que não estão necessariamente associadas aos ótimos da função, uma vez que o movimento dos centroides depende apenas da posição das amostras, não da sua qualidade. Entretanto, ao aplicar o valor proporcional ao valor da função objetivo na amostra como o tamanho do passo, como mostrado em (4.2), um comportamento semelhante é observado: quanto maior o fitness, o maior é o passo dado pelo centroide mais próximo em direção à amostra, o que emula passos frequentemente dados na direção de uma região do espaço de busca.

A equação de ajuste do centroide para o *k-means* modificado para otimização se torna:

$$\mathbf{c}_i = \mathbf{c}_i + \mathbf{peso}_k(\mathbf{x}_k - \mathbf{c}_i) \tag{4.2}$$

onde x_k é o ponto gerado no espaço de busca, c_i é o centroide i,e o *peso* é calculado como segue:

$$\mathbf{peso}_k = \frac{F(\mathbf{x}_k) - \boldsymbol{Pior}}{\boldsymbol{Melhor} - \boldsymbol{Pior}} \tag{4.3}$$

onde $F(\mathbf{x}_k)$ é o valor da função objetivo $F(.)$ no ponto $\mathbf{x}_k$, *Melhor* é o melhor valor da função objetivo entre todas as amostras, e *Pior* é o pior valor da função objetivo entre todas as amostras. Esta expressão é uma proposta heurística baseada em ensaios preliminares.

Assim, se a amostra fica em uma região pouco interessante do espaço de busca, o indivíduo mais próximo sofrerá um pequeno deslocamento, o que ele significa que não é encorajado a explorar essa região. Por outro lado, se a amostra se encontra numa região promissora, o indivíduo irá mover-se significativamente em sua direção.

A estratégia mantém ainda o melhor ponto encontrado, pois, ao final do cálculo dos centroides, o melhor ponto assume a posição do centroide de sua região, caso sua avaliação for melhor do que a avaliação do centroide que o representa. Assim, existe a garantia que o melhor ponto encontrado na população inicial continue como um centroide, caso este possua avaliação melhor do que a do centroide dominante da sua região.

A implementação, naturalmente, está sujeita às dificuldades apresentadas pelo *k-means* original. Com a alteração, uma região com valor da função objetivo mediano e altamente populosa, deverá deslocar um centroide para lá, mesmo não sendo em princípio uma região promissora. Assim, a qualidade da resposta do operador depende da qualidade da amostra e da diferença da avaliação do ponto de ótimo com os demais, ou seja, uma função objetivo com picos bem definidos levará vantagens em relação à função com picos suaves.

Como a proposta pode ser ajustada para qualquer método de clusterização, na próxima seção será descrito sua aplicação no algoritmo *fuzzy c-means*.

4.2.2 Fuzzy C-means

Fuzzy c-means (FCM) é um método de clusterização que utiliza conceitos de conjuntos fuzzy na sua construção, o que viabiliza a pertinência de um objeto a mais de um grupo (Pedrycz & Gomide, 2007). Desta forma, um objeto pode pertencer a várias classes, com graus de pertinência variados. A soma destes graus de pertinência de um objeto para todas as classes deve ser 1, pois o objeto pertence 100% ao universo em que está contido.

O grau de pertinência de um objeto a uma classe assume valores em [0,1]. Assim, em um extremo o objeto pode não pertencer à classe em absoluto (grau de pertinência 0), e, no outro

extremo, pode pertencer exclusivamente à classe (grau de pertinência 1), ou pertencer parcialmente à classe, grau de pertinência entre]0,1[.

O algoritmo *c-means* utiliza uma matriz de partição, que armazena os graus de pertinência dos pontos aos centroides. Inicialmente a matriz de partição é gerada de forma que a soma da coluna seja igual a 1, e a soma da linha seja pelo menos 1. A geração pode ser pseudoaleatória, pois precisa respeitar as restrições de linha e coluna.

Após a geração inicial da matriz, a posição dos centroides é calculada com base na matriz de partição, e, a matriz de partição é calculada com base na posição dos centroides, como apresentado no algoritmo 4.2. Este processo iterativo acontece até a estabilização da matriz de partição, que é um critério canônico de parada do algoritmo (Pedrycz & Gomide, 2007).

Algoritmo 4.2: *Fuzzy c-means*

1) Gere a matriz de pertinência P válida
2) Enquanto o critério de para não for satisfeito
  2.1) **Para** i=1:G faça
    2.1.1) $\mathbf{c}_i(t) = \frac{\sum_{k=1}^{N} (\boldsymbol{P}_{ik})^m(t)\mathbf{x}_k}{\sum_{k=1}^{N}(\boldsymbol{P}_{ik})^m(t)}$
  2.2) **Fim do para**
  2.3) **Para** i=1:G faça
    2.3.1) **Para** k=1:N faça
    2.3.2) $\mathbf{P}_{ik}(t+1) = \frac{1}{\sum_{j=1}^{c}\left(\frac{\|\mathbf{x}_k - \mathbf{c_i}(t)\|}{\|\mathbf{x}_k - \mathbf{c}_j(t)\|}\right)^{2/(m-1)}}$
    2.3.3) **Fim do para**
  2.4) **Fim do para**
3) Fim do enquanto

onde *G* é o número de centroides; *N* é o número de pontos; P_{ik} é o grau de pertinência do ponto *k* no centroide *i*; *m* é o coeficiente de fuzzificação, que molda a função de pertinência presente na matriz de partição.

O desempenho do algoritmo é influenciado pela escolha do número de centroides e de sua posição inicial, bem como pelas propriedades geométricas dos dados. Por conta disto, o algoritmo pode convergir para soluções diferentes em função da posição inicial dos centroides.

Na próxima seção, propõe-se uma modificação junto ao algoritmo c-*means* para que este funcione como um algoritmo de busca.

4.2.2.1 Otimização Baseada no *C-means* Modificado

Seguindo a linha de raciocínio de provocar uma translação do centro de massa para a região promissora, e levando em consideração a abordagem de atualização dos centroides, utilizando a matriz de partição, a proposta consiste em incluir o mesmo peso utilizado no *k-means*, alterando a influência da matriz no posicionamento dos centroides, apenas ajustando a média ponderada já calculada pelo c-*means*. O algoritmo *fuzzy c-means* com a equação modificada para otimização segue no Alg. 4.3.

Algoritmo 4.3: C-means modificado

1) Gere a matriz de pertinência P válida

2) Enquanto o critério de para não for satisfeito

2.1) **Para** i=1:G faça

2.1.1) $$\boldsymbol{c}_i(t) = \frac{\sum_{k=1}^{N} peso_k \, (\boldsymbol{P}_{ik})^m(t)\mathbf{x}_k}{\sum_{k=1}^{N} peso_k \, (\boldsymbol{P}_{ik})^m(t)}$$

2.2) **Fim do para**

2.3) **Para** i=1:G faça

2.3.1) **Para** k=1:N faça

2.3.1.1) $$\mathbf{P}_{ik}(t+1) = \frac{1}{\sum_{j=1}^{c} \left(\frac{\|\mathbf{x}_k - \mathbf{c}_i(t)\|}{\|\mathbf{x}_k - \mathbf{c}_j(t)\|} \right)^{2/(m-1)}}$$

2.3.2) **Fim do para**

2.4) **Fim do para**

3) Fim do enquanto

4) Movimentar o centroide mais próximo do melhor ponto, para a posição deste ponto, se a avaliação do centroide for pior que a avaliação do melhor ponto.

onde G é o número de centroides, N é o número de pontos, P_{ik} é o grau de pertinência do ponto k ao centroide i, m é o coeficiente fuzzificação e $peso_k$ como definido na equação (4.3).

Analisando os centroides calculados pelo *c-means* como um elemento calculado por uma média ponderada dos elementos participantes do grupo, e por consequência, analisando a matriz de pertinência como o peso desta média ponderada; a modificação propõe um novo peso, que promove o deslocamento do centroide para uma região promissora dentro do seu grupo, isto é possível, pois o novo peso é baseado na avaliação do ponto. Sendo assim, o *c-means* modificado será capaz de realizar um processo de busca mantendo a ideia de nichos.

Outro fator é que o peso proveniente da matriz de pertinência visa a proximidade e o peso proveniente da avaliação da função visa a qualidade do ponto. Desta forma, os centroides deverão ser atraídos mais fortemente para os melhores pontos do grupo, de forma mais intensa do que o *k-means*, que sob esta análise teria uma média aritmética, uma vez que a pertinência do ponto ao centroide é *crisp*.

4.3 EXEMPLOS INICIAIS DE OTIMIZAÇÃO POR CLUSTERIZAÇÃO

Nas próximas duas seções, serão apresentados experimentos que têm por objetivo validar a indicação do processo de busca baseado em clusterização. Para isto, busca-se de início aplicar o *k-means* modificado em algoritmos que não enfatizam a manutenção de diversidade da população, e verificar se foi vantajoso utilizar avaliações da função objetivo com o *k-means* modificado.

Optou-se a princípio por problemas bidimensionais, de modo que se possa verificar graficamente os resultados obtidos.

4.3.1 Validação Gráfica da Otimização por Clusterização com o PSO

Para a realização dos experimentos, foram utilizados o *k-means* adaptado para otimização, o PSO e o *k-means* PSO, que consiste na aplicação do *k-means* em 10% das avaliações disponíveis para o PSO, sendo utilizados os centroides encontrados pelo *k-means* como os pontos iniciais do método de inteligência de enxame.

A função utilizada para análise é:

$$F(x, y) = x.sin(4\pi x) - y.sin(4\pi y + \pi) + 1 \quad (4.4)$$

com *x* e *y* definidos no intervalo [-1,1]. Esta função foi escolhida por apresentar comportamento multimodal na região de interesse.

Os algoritmos avaliaram a função objetivo no máximo 1000 vezes. O *k-means* modificado foi aplicado ao espaço de busca com o objetivo de posicionar 30 centroides. A quantidade de centroides se dá pela tentativa de analisar a capacidade do método de encontrar os máximos globais e locais. Na Figura 4.1, é possível verificar que o método encontrou duas regiões com máximos globais, uma pela característica gulosa do método que modifica o centroide mais próximo para o melhor ponto encontrado, a outra região pelo processo de ajuste dos centroides. Além disso, alguns grupos foram disputados por mais de um centroide, e mais de um grupo dominado pelo mesmo centroide. Em ambos os casos, o centroide terminou o processo em uma região menos favorável do que poderia.

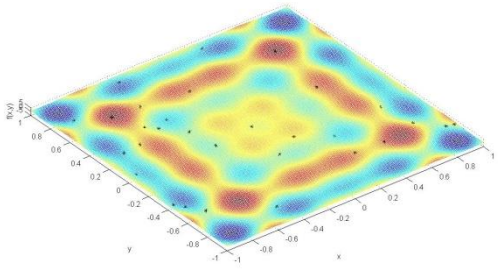


Figura 4.1: Resultado do *k-means* modificado com 30 centroides.

O resultado obtido pela aplicação do PSO canônico está apresentado na Figura 4.2. Neste caso, existe uma tendência das partículas seguirem a melhor partícula obtida, o que está em consonância com a equação de movimento que atualiza a posição dos indivíduos. O experimento retratou o fenômeno, como se pode perceber a partir de uma análise da Figura 4.2, que mostra a melhor posição obtida pelas partículas ao final de sua adaptação.

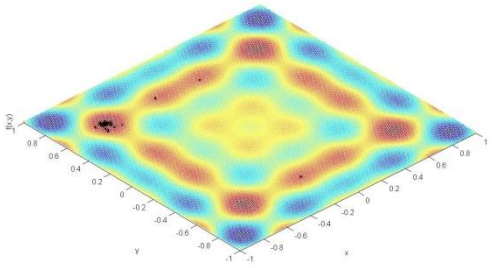


Figura 4.2: Resultado do PSO.

Aplicando-se o *k-means* modificado como pré-processador do PSO, os resultados foram diferentes, como mostra a Figura 4.3. Apesar de utilizar apenas 10 % das avaliações permitidas, o *k-means* modificado promoveu um grande impacto no desempenho do PSO em termos de diversidade.

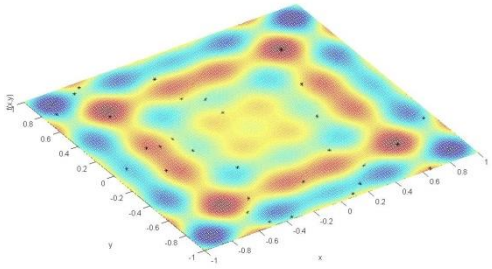


Figura 4.3: Resultado do *k-means* PSO empregando 1000 avaliações.

Apesar disso, fica claro que o algoritmo ainda não convergiu, por conta do espalhamento das partículas. Para que se tivesse análise mais efetiva da convergência, aumentou-se a quantidade de avaliações para dez mil, e os resultados estão apresentados na Figura 4.4. Neste caso, nota-se que os quatro ótimos globais da função objetivo são encontrados, graças à diversidade gerada através do *k-means* modificado.

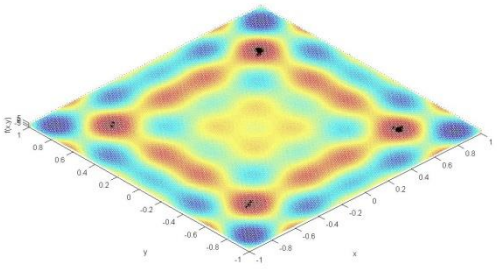


Figura 4.4: *K-means* PSO empregando 10000 avaliações.

A metodologia proposta conseguiu posicionar pontos em regiões promissoras. Este posicionamento depende do número de centroides escolhidos, da quantidade de ótimos locais da função mapeada e da qualidade do mapeamento da função, ou seja, da quantidade de pontos que servirão de base para a aplicação da metodologia e do espalhamento destes pontos.

Neste experimento, a metodologia aplicada por intermédio do *k-means* modificado promoveu melhoria na qualidade das respostas do PSO. A metodologia não possui uma estratégia de refinamento, mas encontrou de forma satisfatória as regiões promissoras do problema proposto. Vale lembrar que o PSO não perdeu suas características, e obteve os resultados apresentados graças aos pontos iniciais gerados pelo *k-means* modificado. Entretanto, continuando o aumento de avaliações, a tendência do PSO é a perda da diversidade.

Na próxima seção, o *k-means* modificado será testado como pré-processador do AG.

4.3.2 Validação Gráfica da Otimização por Clusterização com o AG

Para este experimento, o limite de avaliações da função objetivo foi 300, e, destas, 150 avaliações foram utilizadas para aplicar o *k-means* modificado. O *k-means* modificado gerou 30 centroides e o AG foi configurado com uma população de 30 indivíduos, utilizando os resultados do *k-means* modificado como população inicial. Duas funções foram utilizadas, sendo a primeira a mesma do experimento anterior:

$$Fa(x,y) = x\,sin(4\pi x) - y\,sin(4\pi y + \pi) + 1 \tag{4.5}$$

onde x, y ∈ [-2,2].

A segunda função foi:

$$Fb(x,y) = 0.5 - \left(\frac{sin\left(\sqrt{x^2+y^2}\right)^2 - 0.5}{0.001(x^2+y^2)+1}\right) \tag{4.6}$$

onde x, y ∈ [-10,10].

A Fig. 4.5 apresenta a quantidade de ótimos locais existentes dentro do espaço de busca da função Fa; já a Fig. 4.6 apresenta o resultado da distribuição dos pontos iniciais na projeção 2D da referida função: este é o estado inicial do AG.

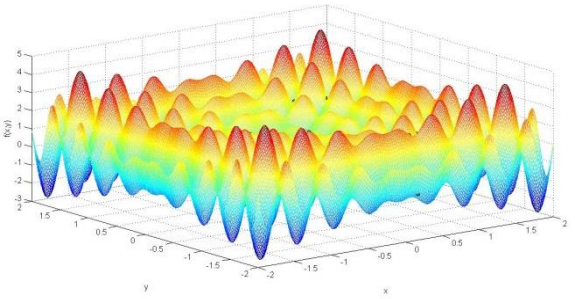


Figura 4.5: Estado inicial do AG aplicado à função Fa.

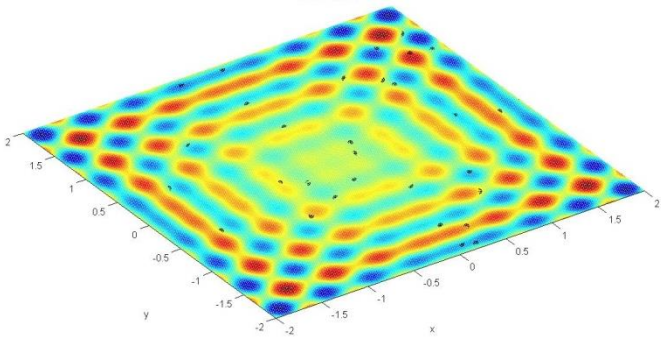


Figura 4.6: Estado inicial do AG aplicado à função Fa.

Na Figura 4.7, nota-se que o AG encontrou apenas um máximo global, e perdeu rapidamente sua diversidade.

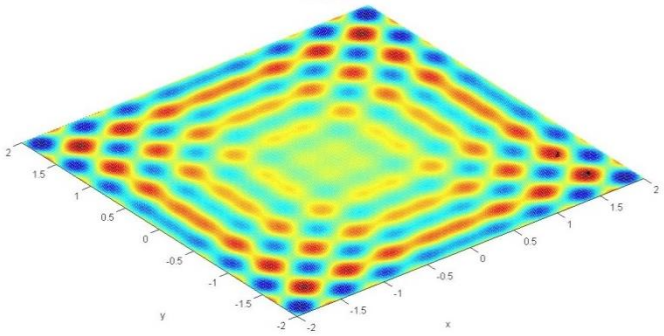


Figura 4.7: Resultados obtido pelo AG aplicado à função Fa.

Nas Figs. 4.8 e 4.9, a metodologia proposta será testada por meio do uso do *k-means* modificado como pré-processador. Desta forma, a modificação proposta será responsável pela geração dos pontos iniciais do AG.

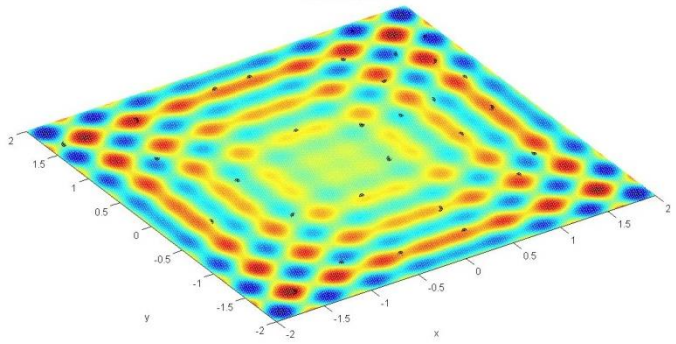


Figura 4.8: Estado inicial do AG com *k-means* modificado como pré-processador

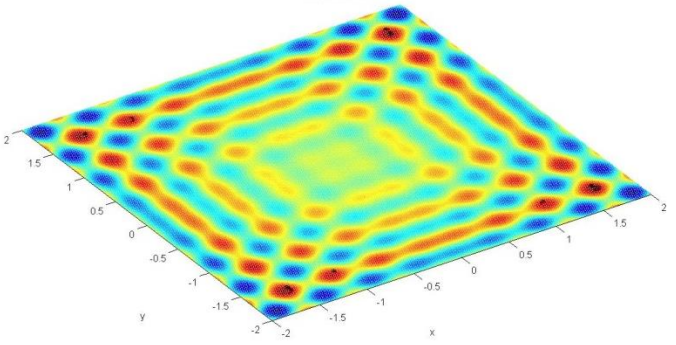


Figura 4.9: Resultados obtidos pelo AG com *k-means* modificado como pré-processador aplicado à função Fa.

Como apresentado na Figura 4.8, a aplicação do *k-means* modificado posicionou indivíduos em regiões promissoras, muito próximas aos máximos globais. Estes indivíduos evoluíram posteriormente no caminho do ótimo, e, por conta da perda de diversidade do algoritmo, grupos de indivíduos foram posicionados nas regiões dos quatro ótimos globais. O pré-processamento introduziu diversidade no algoritmo, mas o algoritmo não perdeu suas características intrínsecas. Desta forma, havendo um aumento de avaliações, a perda de diversidade poderia se intensificar.

Visando aplicar a metodologia em uma função com apenas um máximo global e com complexa transição entre os máximos locais para o máximo global, são mostrados a seguir os resultados do experimento com a função Fb.

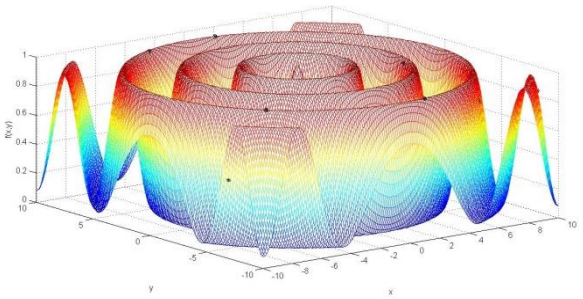


Figura 4.10: Estado inicial do AG aplicado sobre a função Fb.

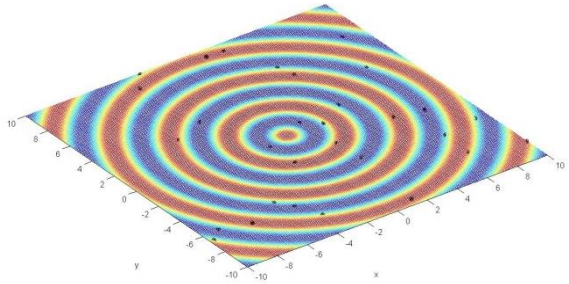


Figura 4.11: Estado inicial do AG aplicado sobre a função Fb.

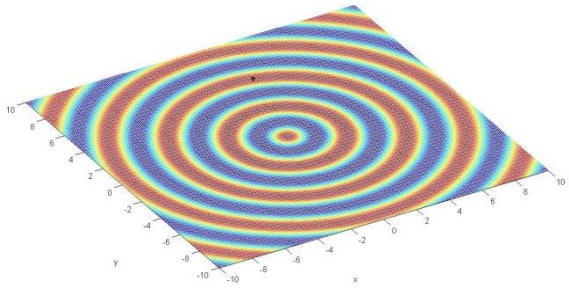


Figura 4.12: Resultados obtidos pelo AG aplicado à função Fb.

Note na Figura 4.12 que, a partir do estado inicial representado nas Figs. 4.10 e 4.11, o AG perdeu rapidamente sua diversidade, e ficou preso em um máximo local.

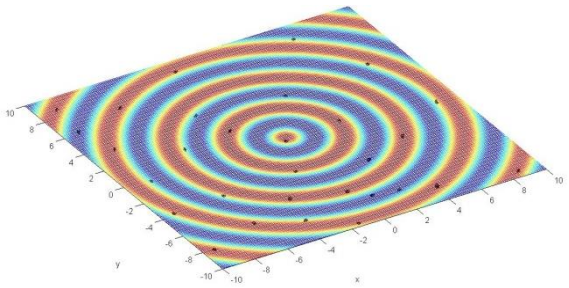


Figura 4.13: Estado inicial do AG após aplicação do *k-means* modificado.

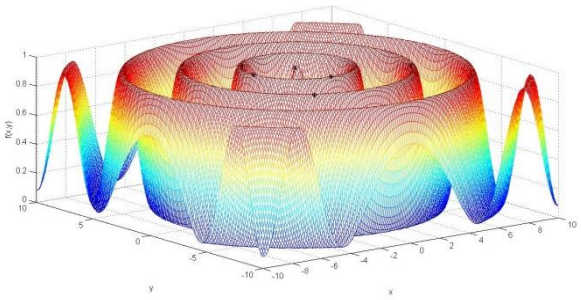


Figura 4.14: Resultados obtidos pelo AG aplicado à função Fb, com *k-means* modificado como pré-processador.

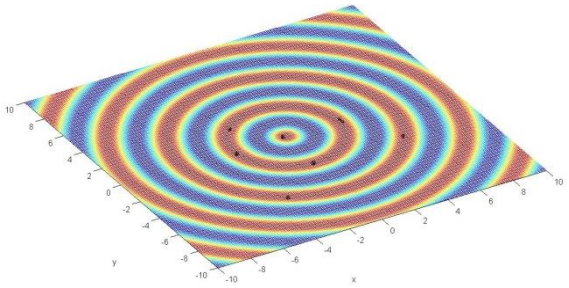


Figura 4.15: Resultados obtidos pelo AG aplicado à função Fb, com *k-means* modificado como pré-processador.

Por outro lado, como mostram as Figs. 4.13 a 4.15, o *k-means* modificado favoreceu a manutenção dos pontos nas regiões promissoras, gerando um bom estado inicial ao AG, que possibilitou o encontro do máximo global.

Uma característica do processo de busca baseado em clusterização é a capacidade de promover diversidade sem a necessidade de aumentar a quantidade de avaliações da função objetivo. O método proposto mostrou-se eficaz nos problemas em duas dimensões que foram estudados, e

promoveu o aumento de diversidade. Na próxima seção, problemas mais desafiadores serão tratados.

4.4 EXPERIMENTOS

Com o objetivo de analisar o processo de busca baseado em clusterização, o método *k-means* modificado foi utilizado como pré-processador de algoritmos de otimização. Cinco experimentos mais complexos foram realizados, e, sempre que possível, os parâmetros de todos os algoritmos foram os mesmos. Desta forma, nenhum algoritmo seria privilegiado em relação aos demais.

Para a obtenção dos resultados foram realizadas 30 execuções de cada algoritmo em cada função, e o melhor resultado encontrado em cada execução foi utilizado para o cálculo da média e do desvio padrão. Com o objetivo de evitar que o *k-means* modificado não convirja adequadamente, o critério de parada foi fixado em 20 iterações para atualização dos centroides. O algoritmo também para quando os centroides convergiram.

4.4.1 Experimento 1

Neste experimento, o *k-means* modificado foi utilizado como pré-processador do AG, CLONALG e PSO, algoritmos já discutidos nesta tese, nas seções 2.2.4.1, 2.2.3.1 e 2.2.5.1, respectivamente. O *k-means* modificado como pré-processador utilizou 30% das avaliações da função objetivo para criar sua base de dados. Os centroides obtidos pelo *k-means* modificado foram utilizados como população inicial dos algoritmos, e neste caso, eles foram chamados de KAG, KCLONALG e KPSO, respectivamente. Além disso, as performances dos algoritmos com e sem o *k-means* modificado foram comparadas.

Seis funções de *benchmarks* de otimização foram utilizadas neste experimento e, a função F1 foi selecionada por não possuir mínimos locais, apenas um mínimo global. Todas as funções estão descritas no apêndice A.

Os principais parâmetros dos experimentos foram:

- Número de indivíduos ou partículas = 30
- Número de descendentes = 10
- Dimensão = 30
- Número de avaliações da função objetivo = 1.000

Tabela 4.1: Valores da média e do desvio padrão calculados com o melhor resultado encontrado em cada uma das 30 execuções dos algoritmos – espaço de busca com 30 dimensões, e 1.000 avaliações da função objetivo em cada execução.

Funções	Algoritmos						
	Global	AG	KAG	Clonalg	KClonalg	PSO	KPSO
Esfera	0,00	18404±3834	14771±3200	43487±6281	30853±3501	6295±2226	**4635±1166**
Rosenbrock	0,00	3.63E9± 1.52E9	2.63E9±1.03E9	1.22E10±2.62E9	7.15E9±1.47E9	3.99E8±3.00E8	**2.59E8±2.41E8**
Griewank	0,00	168.07±37.71	130.03±30.70	571.83±46.24	390.06±38.22	63.27±15.39	**43.84±15.54**
Schwefel	-12569,49	-6577±557	**-6610±557**	-3646±500	-4000±420	-3803±598	-3794±510
Rastrigin	0,00	215.15±24.89	**191.72±22.65**	214.41±23.42	205.55±22.84	237.31±21.98	221.14±24.05
F1	0,00	93.62±12.22	92.74±9.97	**90.93±19.33**	101.84±12.07	151.32±19.78	139.56±16.89
Schaffer	1,00	0.69±0.03	0.73±0.02	0.65±0.01	0.71±0.02	0.88±0.02	**0.92±0.01**

A aplicação do *k-means* modificado como pré-processador melhorou o desempenho de todos os algoritmos, com exceção do CLONALG aplicado na função F1. Esta função é unimodal, e o aumento da diversidade não provocou melhora nos resultados obtidos. O teste *t de Student* foi aplicado sobre os resultados apresentados, obtendo-se h=0 e p=1 em todos os casos. Isto indica que os resultados apresentaram confiabilidade estatística - os intervalos de confiança estão apresentados na Tabela 4.2.

Tabela 4.2: O intervalo da média com 95% de confiança definidos pelo teste t, aplicado sobre os resultados dos algoritmos.

Funções	Algoritmos					
	AG	KAG	Clonalg	KClonalg	PSO	KPSO
Esfera	[1.69E4, 1.98E4]	[1.35E4, 1.59E4]	[4.11E4, 4.58E4]	[2.95E4, 3.21E4]	[5.46E3, 7.12E3]	[4.2E3, 5.07E3]
Rosenbrock	[3.06E9, 4.20E9]	[2.24E9, 3.01E9]	[1.12E10, 1.31E10]	[6.60E9, 7.71E9]	[2.87E8, 5.12E8]	[1.78E8, 3.38E8]
Griewank	[153.99, 182.15]	[118.56, 141.49]	[554.56, 589.10]	[375.79, 404.33]	[57.52, 69.02]	[38.04, 49.65]
Schwefel	[-6786.00,-6369.92]	[-6818.39,-6401.87]	[-3833.40, -3459.80]	[-4157.39, -3843.18]	[-4026.86, -3579.89]	[-3984.79, -3603.65]
Rastrigin	[205.86, 224.45]	[183.26, 200.18]	[205.66, 223.16]	[197.02, 214.08]	[229.11, 245.52]	[212.16, 230.12]
F1	[89.06, 98.18]	[89.02, 96.47]	[83.71, 98.15]	[97.33, 106.35]	[143.93, 158.71]	[133.25, 145.86]
Schaffer	[0.68, 0.70]	[0.73, 0.74]	[0.65, 0.66]	[0.71, 0.72]	[0.87, 0.89]	[0.91, 0.92]

O tempo de execução é um fator importante na comparação de algoritmos de otimização - por isso, as próximas figuras apresentam os resultados obtidos e os tempos utilizados na resolução.

O uso do *k-means* modificado como pré-processador não aumentou significativamente o tempo de resposta dos algoritmos, causando pequena melhora no caso do AG e produziu o pior resultado no caso do PSO, como mostra o *boxplot* da Figura 4.16b. Além disso, o uso do *k-means* modificado, na mediana, melhorou os resultados de todos os algoritmos.

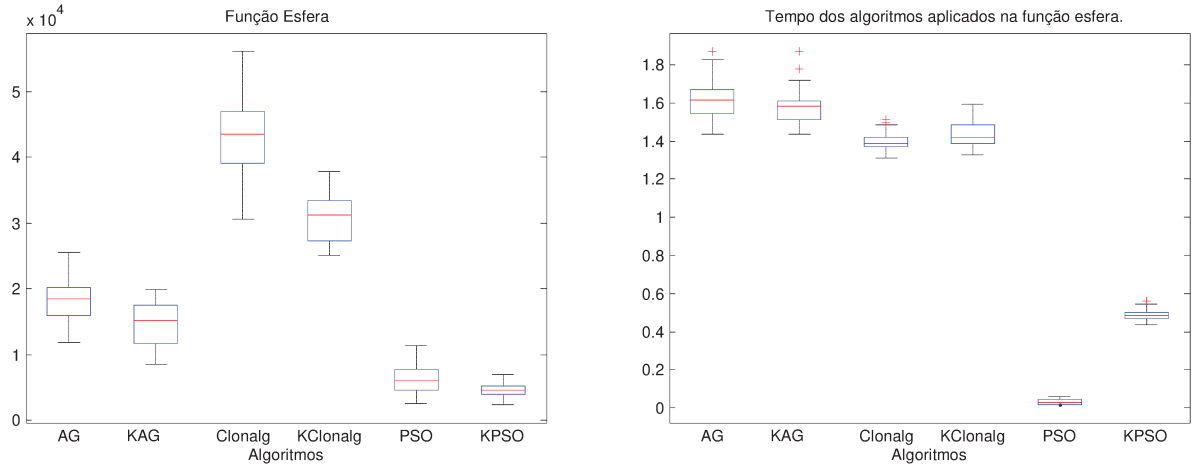
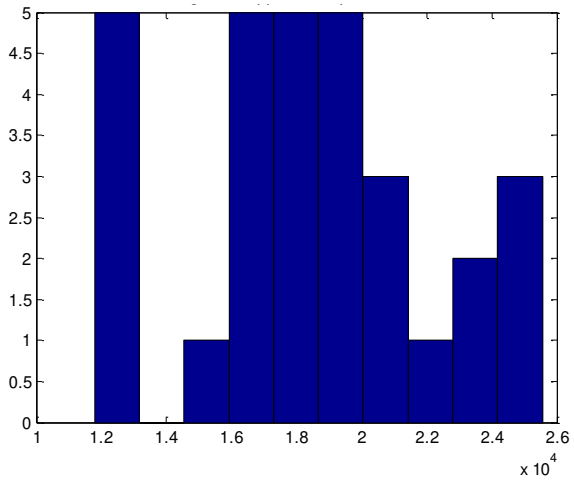


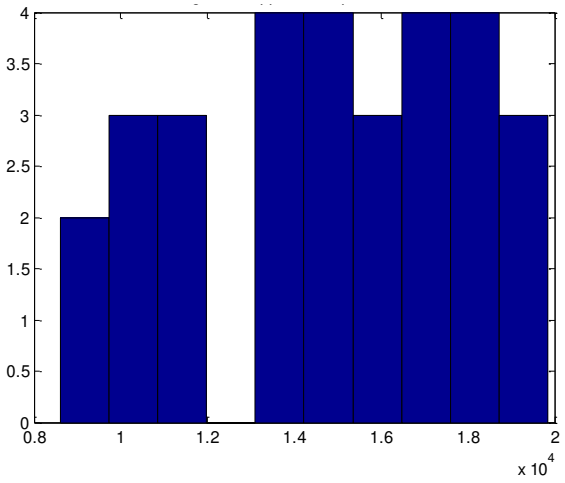
Figura 4.16 a) Boxplot com os resultados dos algoritmos aplicados na função esfera. b) Boxplot com os tempos dos algoritmos aplicados na função esfera.

Os resultados apresentados nas tabelas informam a média é o desvio padrão do melhor resultado encontrado em cada execução dos algoritmos, mas não apresentam a distribuição destes resultados. Um resultado muito ruim pode deslocar a média, e neste caso, a exceção prejudica o caso geral, e a comparação entre os algoritmos perde veracidade, por este motivo alguns histogramas serão apresentados na sequência.

Nos histogramas das Figuras 4.17, 4.18 e 4.19, o uso do *k-means* modificado provocou um translado das soluções, causando melhoria nas execuções individuais, a ponto de o eixo *x* alterar o seu limite.



a) Histograma do AG aplicado na função esfera.



b) Histograma do KAG aplicado na função esfera.

Figura 4.17:

Com CLONALG, como mostrado na Fig. 4.18, as melhorias foram ainda mais significativas.

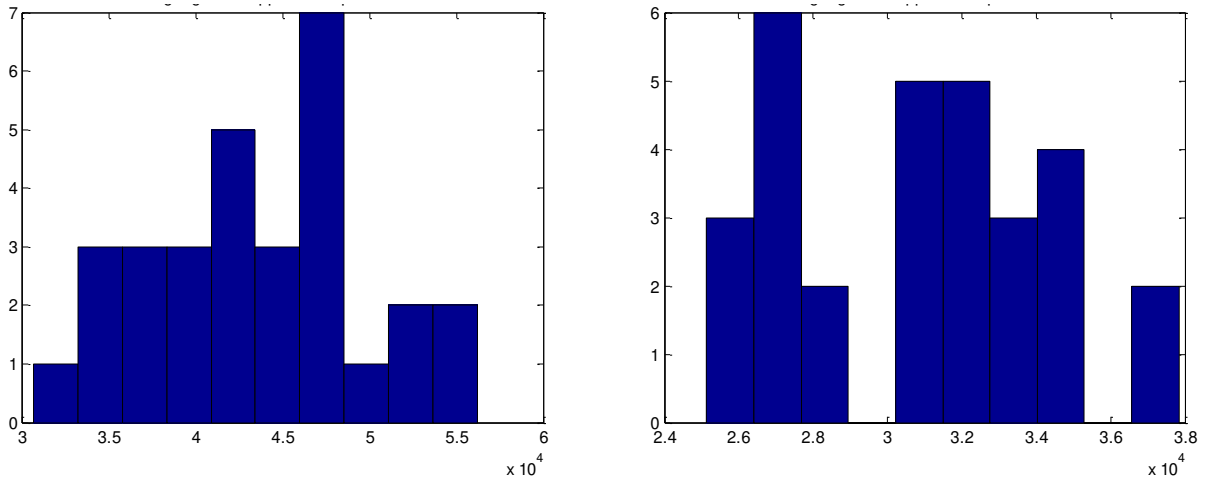


Figura 4.18: a) Histograma do CLONALG aplicado na função esfera b) Histograma do KCLONALG aplicado na função esfera.

No caso do PSO, o limite do boxplot da Figura 4.19a é de 12000, enquanto para o KPSO é de 8000.

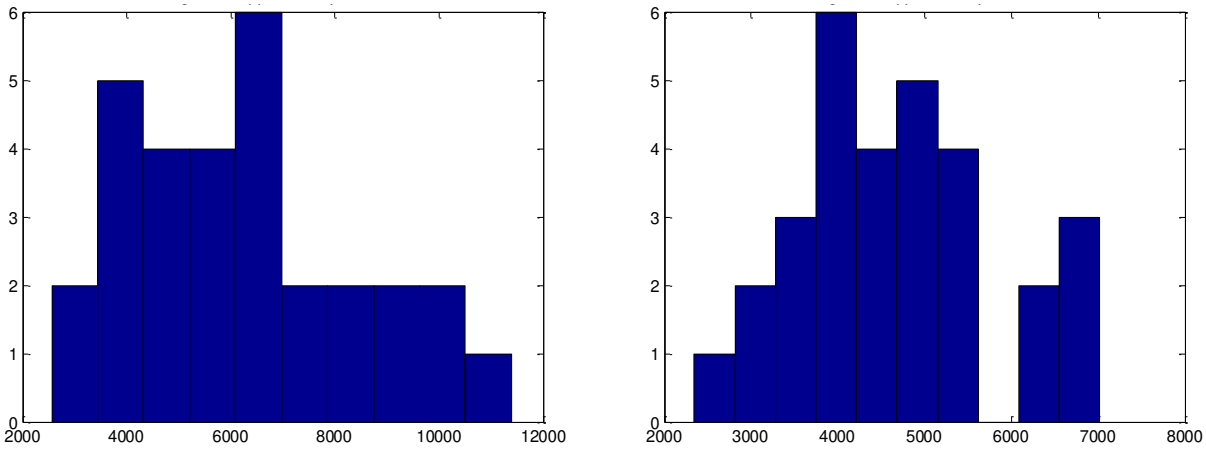


Figura 4.19 a:) Histograma do PSO aplicado na função esfera. b) Histograma do KPSO aplicado na função esfera.

Em suma, em todos os histogramas, o uso do *k-means* modificado como pré-processador melhorou os resultados finais.

Nos próximos *boxplots*, serão apresentados os resultados dos algoritmos e seu tempo de execução. Nas funções *Rosenbrock, Griewank, Schwefel, Rastrigin*, de forma geral, o *k-means* modificado promoveu melhora nos resultados, gerando atraso somente na execução do PSO. Aqui cabe uma ressalva, pois o PSO foi implementado no Matlab®, e de forma matricial, enquan-

to o *k-means* modificado, também implementado no Matlab®, mas de forma iterativa. É sabido que o Matlab® possui alta capacidade de otimização para execução de operações matriciais, portanto conclui-se que esta seja a origem da diferença significativa de tempo do PSO.

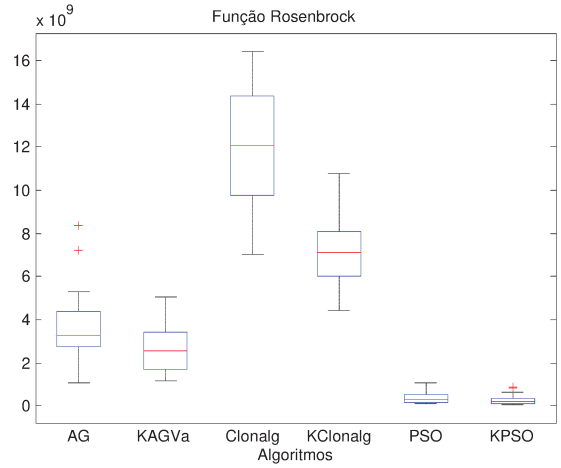


Figura 4.20: Boxplot dos resultados dos algoritmos aplicados na função Rosenbrock.

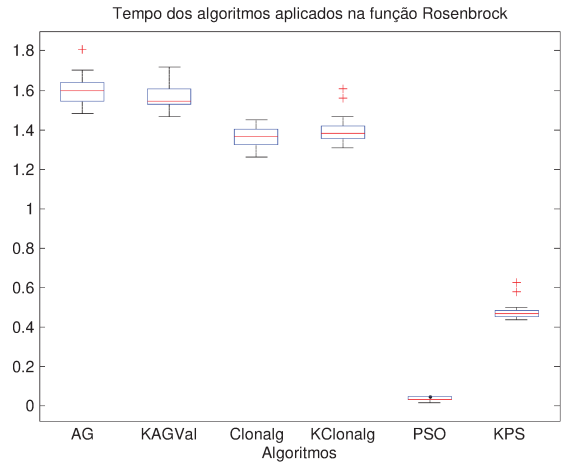


Figura 4.21: Boxplot dos tempos dos algoritmos aplicados na função Rosenbrock.

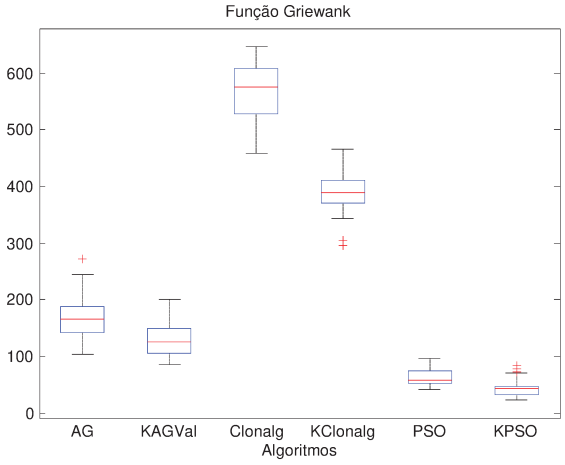
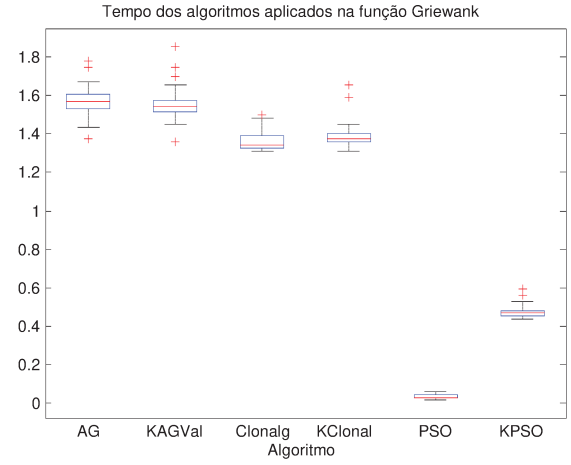
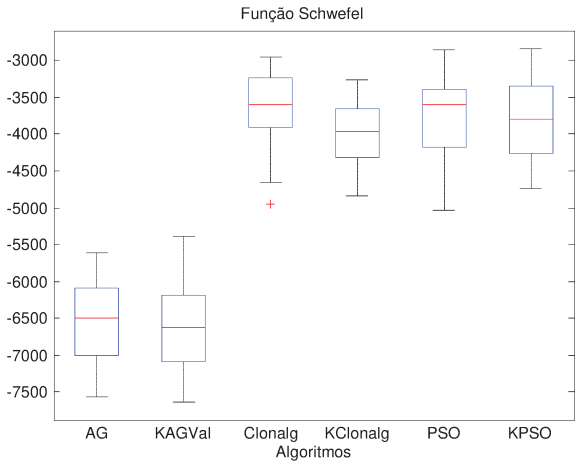


Figura 4.22 a) Boxplot dos resultados dos algoritmos aplicados na função Griewank.



b) Boxplot com os tempos dos algoritmos aplicados na função Griewank.



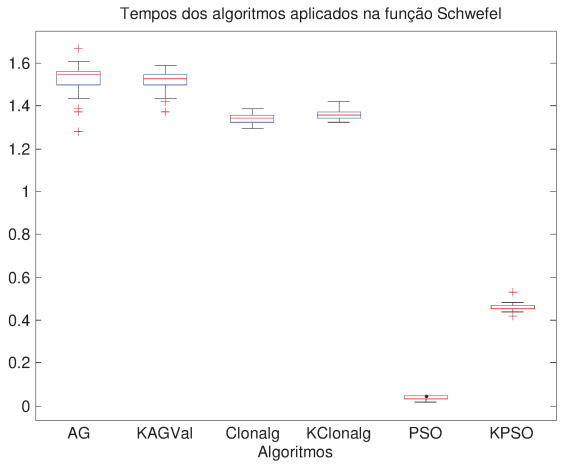
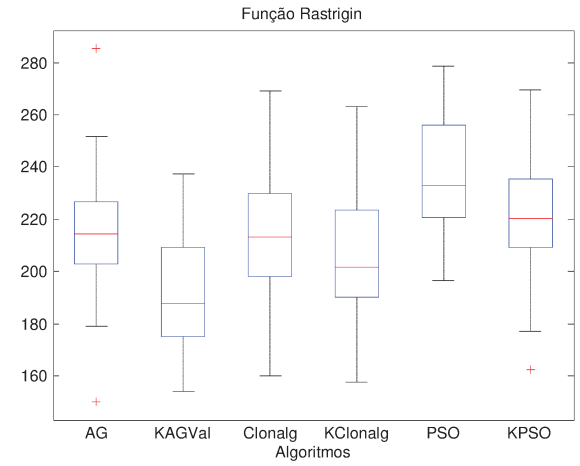


Figura 4.23 a) Boxplot dos resultados dos algoritmos aplicados na função Schwefel.

b) Boxplot com os tempos dos algoritmos aplicados na função Schwefel.



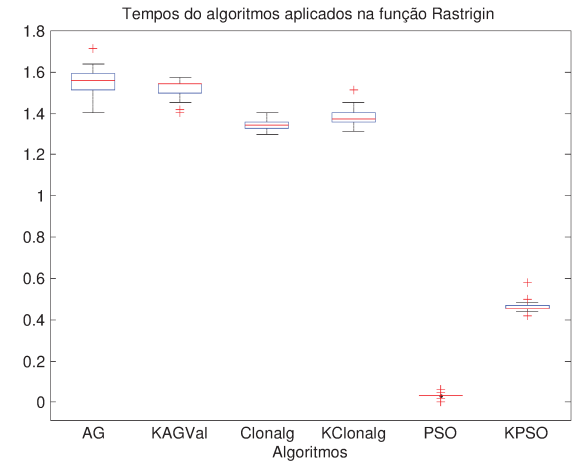
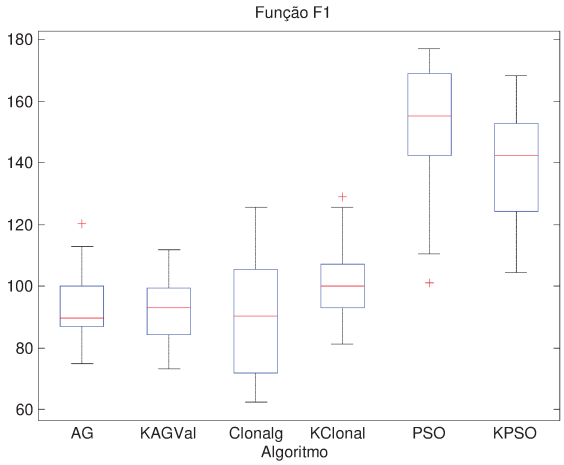


Figura 4.24 a) Boxplot com os resultados dos algoritmos aplicados na função Rastrigin.

b) Boxplot com os tempos dos algoritmos aplicados na função Rastrigin.

Na função F1, o uso do *k-means* modificado não promoveu melhora da mediana para o AG e o CLONALG: isto mostra que a introdução de diversidade não provocou melhora de forma geral em uma função estritamente unimodal.



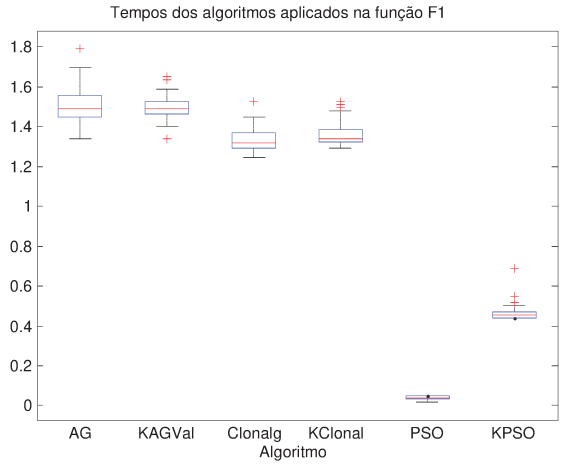
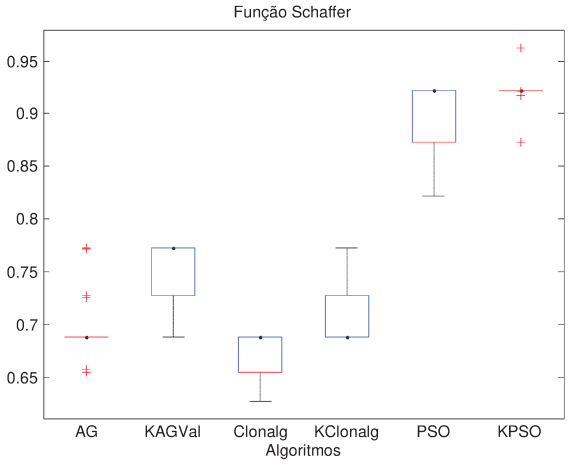


Figura 4.25 a) Boxplot com os resultados dos algoritmos aplicados na função F1.

b) Boxplot com os tempos algoritmos aplicados na função F1.

Na função *Schaffer*, os melhores resultados estão mais próximos do um, e, portanto, mais altos no bloxplot, como mostrado a seguir.



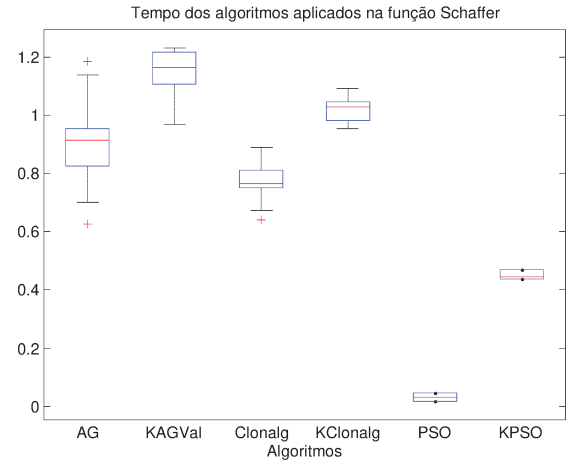


Figura 4.26: Boxplot com os resultados dos algoritmos aplicados na função Schaffer.

Figura 4.27: Boxplot com os tempos do algoritmos aplicados na função Schaffer.

O uso da metodologia de busca baseada em clusterização neste experimento introduziu, nitidamente, diversidade. Quando esta diversidade foi útil no processo de busca, os resultados obtidos melhoraram.

4.4.2 Experimento 2

Neste experimento, o *k-means* modificado foi aplicado com os algoritmos AG, CLONALG e PSO, no início e no meio da execução dos algoritmos. Assim, o *k-means* modificado foi utilizado como pré-processador e como operador de manutenção de diversidade.

Para aplicar o *k-means* modificado como operador de manutenção de diversidade, foram utilizados os pontos gerados até a metade das avaliações disponíveis da função objetivo: os indivíduos com avaliação pior que a média populacional foram substituídos pelos centroides obtidos. Entretanto, caso todos os indivíduos tivessem a mesma avaliação, metade da população seria aleatoriamente substituída.

Assim, o *k-means* modificado foi aplicado duas vezes. A primeira, no início, para a geração da população inicial, e a segunda substituindo os pontos estagnados ou pontos com avaliação ruim na metade da execução do algoritmo. Todas as aplicações do *k-means* modificado foram propostas para melhorar a diversidade do processo. As configurações foram as mesmas do experimento anterior.

Comparando a Tabela 4.3 com a Tabela 4.1, é possível perceber que a introdução do operador de diversidade ao longo da execução dos algoritmos não provocou alterações significativas. No caso da função esfera houve mesmo uma piora nos resultados; no caso da função *Rosenbrock*, houve uma melhora sutil.

Tabela 4.3: Média e desvio padrão calculados com os melhores valores da função objetivo– Espaço de busca com 30 dimensões, e 1.000 avaliações da função objetivo.

Função	Algoritmos			
	Global	KAG	KClonalg	KPSO
Esfera	0,00	15031±3608	32243±2907	**6399±3323**
Rosenbrock	0,00	2.25E9±8.40E8	6.80E9±1.47E9	**3.30E8±3.16E8**
Griewank	0,00	127.38±21.32	356.77±40.91	**50.62±24.39**
Schwefel	-12569,49	**-6419±523**	-4185±441	-3836±470
Rastrigin	0,00	**189.93±20.74**	211.19±21.14	232.16±25.13
F1	0,00	**93.16±12.77**	100.82±10.91	136.98±14.32
Schaffer	1,00	0.75±0.03	0.72±0.03	**0.91±0.01**

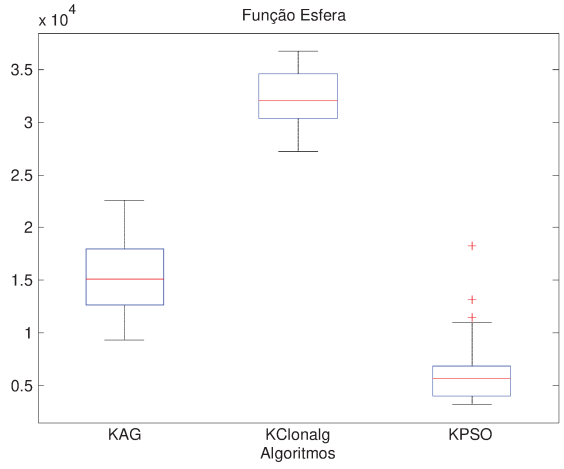
O teste t de *Student* foi aplicado sobre os resultados, e obteve h=0 e p=1, o que garante a

significância dos experimentos.

Tabela 4.4: O intervalo da média com 95% de confiança definidos pelo teste t, aplicado sobre os resultados dos algoritmos.

Funções	Algoritmos		
	KAG	KClonalg	KPSO
Esfera	[13683.81,16378.36]	[31158.22,33329.31]	[5158.45,7640.36]
Rosenbrock	[1.94E9,2.57E9]	[6.25E9,7.35E9]	[2.12E8,4.48E8]
Griewank	[119.42,135.35]	[341.49,372.04]	[41.51,59.73]
Schwefel	[-6614.66,-6223.89]	[-4350.44,-4020.71]	[-4012.23,-3660.72]
Rastrigin	[182.18,197.68]	[203.29,219.08]	[222.78,241.55]
F1	[88.39,97.93]	[96.75,104.90]	[131.63,142.33]
Schaffer	[0.74,0.77]	[0.71,0.73]	[0.90,0.92]

Nos próximo boxplots, apresentados nas figuras 4.28 até 4.34, estão os resultados obtidos e os tempos de execução. Em relação ao experimento 1, a introdução do *k-means* modificado durante a execução dos algoritmos não provocou alterações significativas nos resultados, as distribuições e proporções foram mantidas.



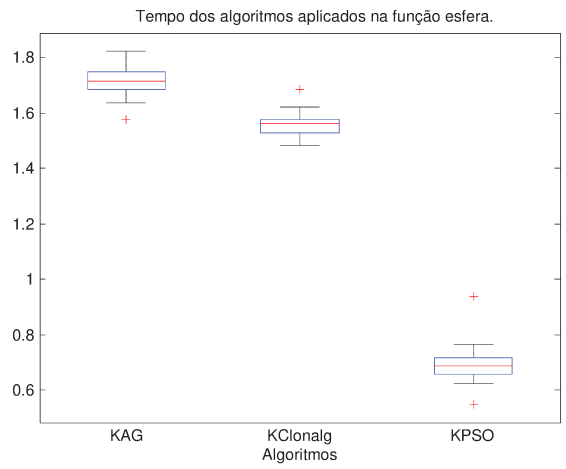


Figura 4.28 a) Boxplot com os resultados dos algoritmos aplicados na função esfera. b) Boxplot com os tempos dos algoritmos aplicados na função esfera.

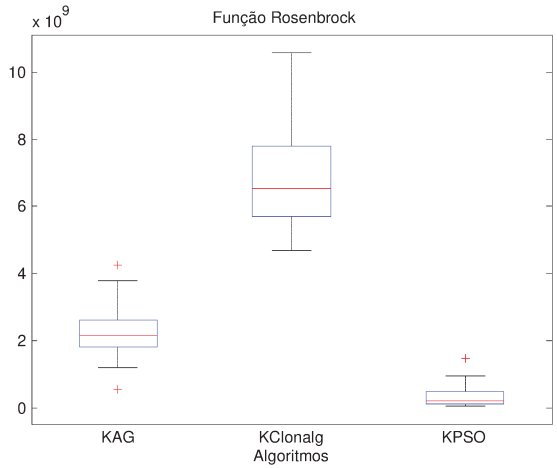
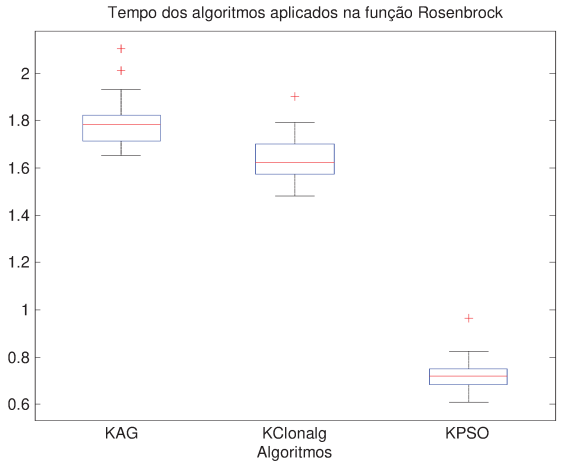


Figura 4.29 a) Boxplot com os resultados dos algoritmos aplicados na função Rosenbrock.



b) Boxplot com os tempos dos algoritmos aplicados na função Rosenbrock.

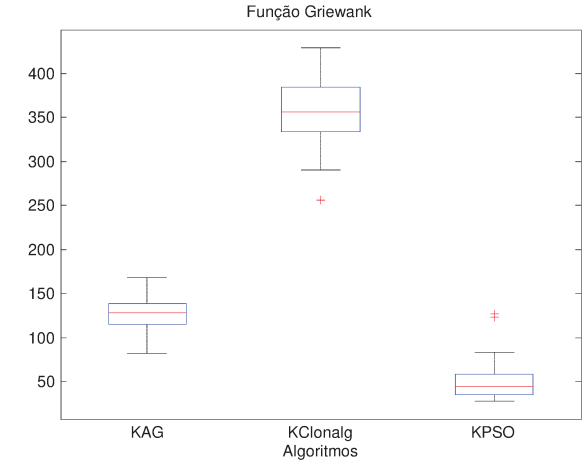
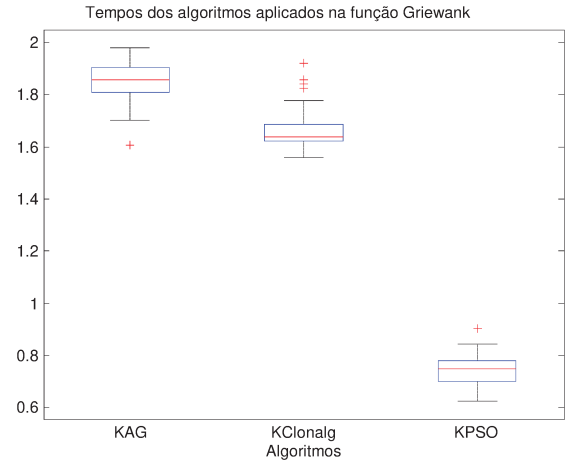
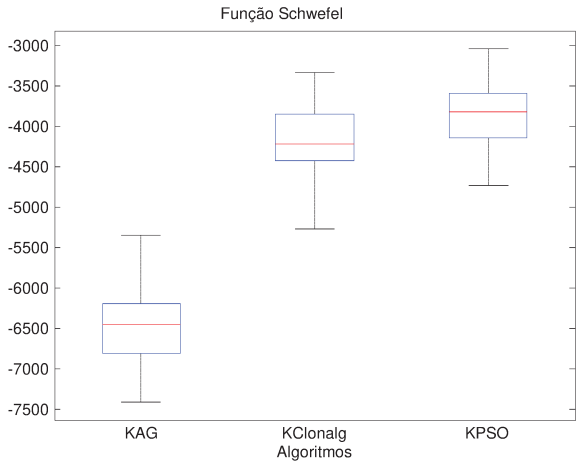


Figura 4.30 a) Boxplot com os resultados de algoritmos aplicados na função Griewank.



b) Boxplot com os tempos dos algoritmos aplicados na função Griewank.



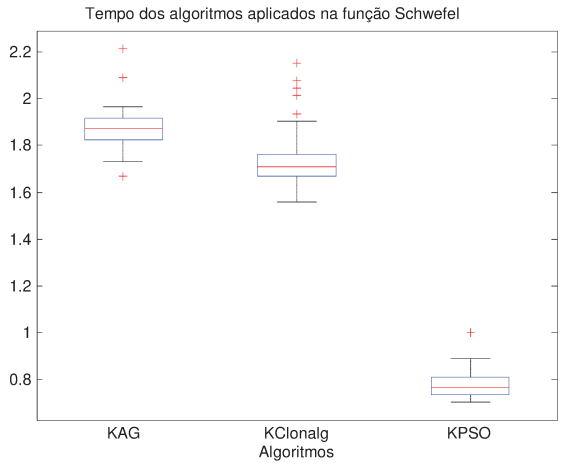
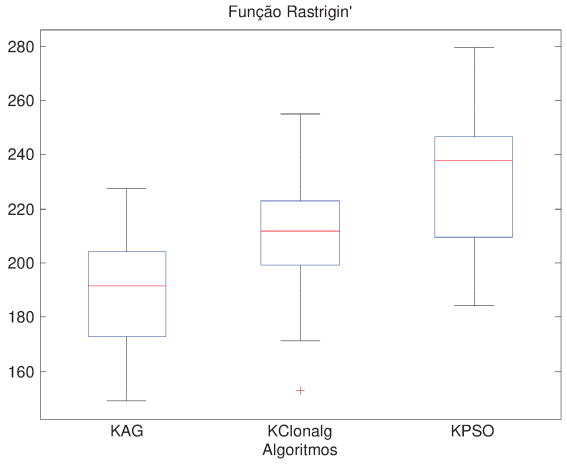


Figura 4.31 a) Boxplot com os resultados dos algoritmos aplicados na função Schwefel.

b) Boxplot com os tempos dos algoritmos aplicados na função Schwefel.



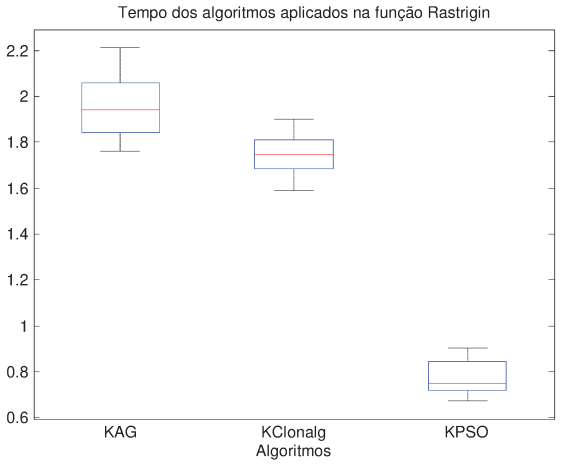
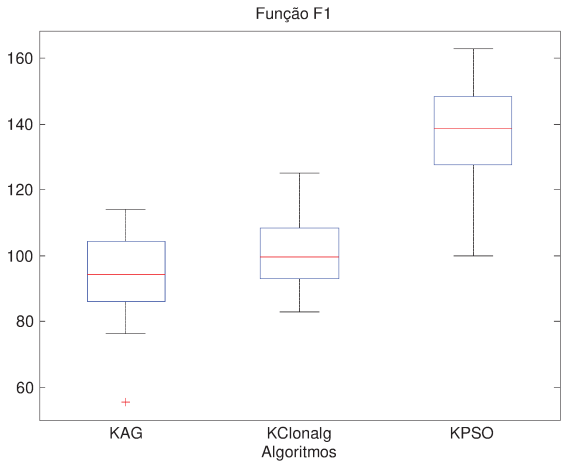


Figura 4.32 a) Boxplot com os resultados dos algoritmos aplicados na função Rastrigin.

b) Boxplot com os tempos dos algoritmos aplicados na função Rastrigin.



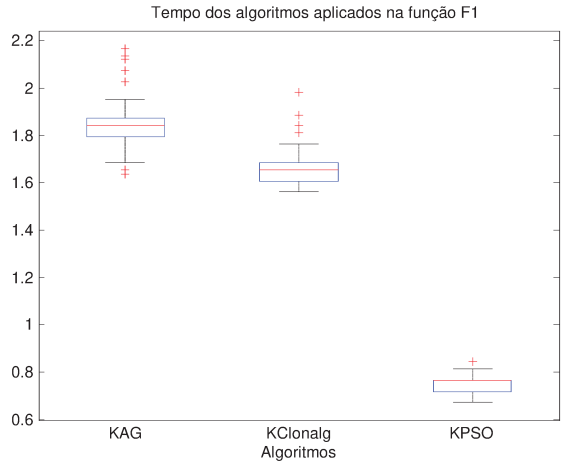
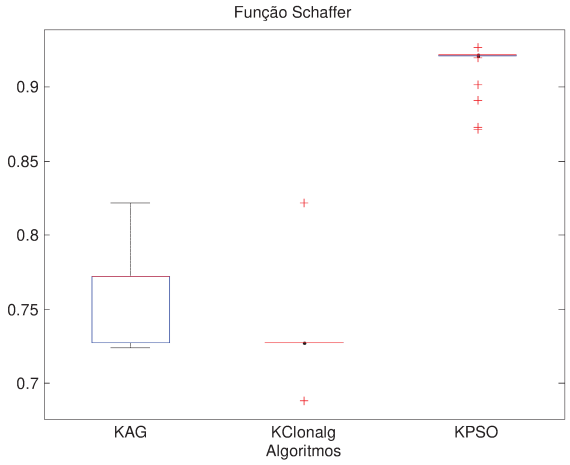


Figura 4.33 a) Boxplot com os resultados dos algoritmos aplicados na função F1.

b) Boxplot com os tempos dos algoritmos aplicados na função F1.



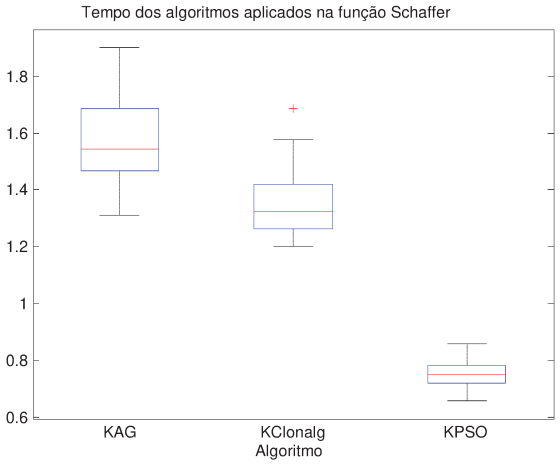


Figura 4.34 a) Boxplot com os resultados dos algoritmos aplicados na função Schaffer.

b) Boxplot com os tempos dos algoritmos aplicados na função Schaffer.

Este experimento sugere que a introdução de diversidade durante a execução dos algoritmos não é necessariamente essencial, desde que esta diversidade já tenha sido introduzida no

início. Isto deu suporte à ideia de que um processo de busca deve possuir grande diversidade inicial, e caminhar para o refinamento ao longo de sua execução.

Na próxima seção, o experimento avaliará algoritmos que utilizam algum tipo de raio como elemento do seu processo de busca, e que possuem estratégias de manutenção de diversidade.

4.4.3 Experimento 3

Este experimento consiste em aplicar as metodologias *Fitness Sharing*, *Opt-aiNet* e *Clearing* nas mesmas sete funções do experimento anterior. O objetivo é comparar o uso do *k-means* modificado com técnicas que utilizam raios para definir nichos, e promovem, desta forma, a manutenção da diversidade ao longo de suas execuções.

Na Tabela 4.5, o algoritmo AG com *Clearing* obteve o melhor resultado em todas as funções, exceto para a função *Schaffer*: nesta, o AG com *Fitness Sharing* foi melhor. Entretanto, Em comparação com a Tabela 4.1, os resultados foram piores para as funções Esfera, *Rosenbrock, Griewank, Rastringin* e *Schaffer*, apresentando resultado ligeiramente melhor nas funções *Schwefel* e F1.

Tabela 4.5: Média e desvio padrão do melhor de cada uma das 30 execuções dos algoritmos – Espaço de busca com 30 dimensões, 1.000 avaliações da função objetivo.

Funções	Algoritmos			
	Global	AG Clearing	AG Fitness Sharing	Opt-aiNet
Esfera	0,00	**16823±2471**	48069±6535	39605±5210
Rosenbrock	0,00	**3.67E9±1.39E9**	1.77E10±4.43E9	1.26E10±2.22E9
Griewank	0,00	**153.43±37.59**	440.84±63.69	545.36±53.88
Schwefel	-12569,49	**-6897±655**	-6638±707	-4292±395
Rastrigin	0,00	**202.25±20.32**	366.57±26.39	358.40±19.06
F1	0,00	**90.40±9.51**	180.87±19.26	184.11±12.41
Schaffer	1,00	0.68±0.03	**0.69±0.02**	0.68±0.02

O teste t de *Student* foi aplicado aos resultados, e obteve h =0 e p=1 para todos os resultados.

Tabela 4.6: O intervalo com 95% confiança para a média, definido pelo teste t aplicado aos resultados dos algoritmos.

Funções	Algoritmos		
	AG Clearing	AG Fitness Sharing	Opt-aiNet
Sphere	[15900.31,17746.15]	[45628.94,50509.57]	[37659.91,41551.55]
Rosenbrock	[3.15E8,4.19E9]	[1.61E10,1.94E10]	[1.17E10,1.34E10]
Griewank	[139.39,167.47]	[417.06,464.63]	[525.24,565.48]
Schwefel	[-7142.41,-6652.80]	[-6902.10,-6373.95]	[-4439.83,-4144.72]
Rastrigin	[194.67,209.84]	[356.71,376.42]	[351.28,365.52]
F1	[86.84,93.95]	[173.68,188.07]	[179.48,188.75]
Schaffer	[0.67,0.70]	[0.68,0.69]	[0.67,0.69]

Os próximos boxplots apresentam os melhores resultados obtidos pelo AG *Clearing*, *Fitness Sharing* e *Opt-aiNet*.

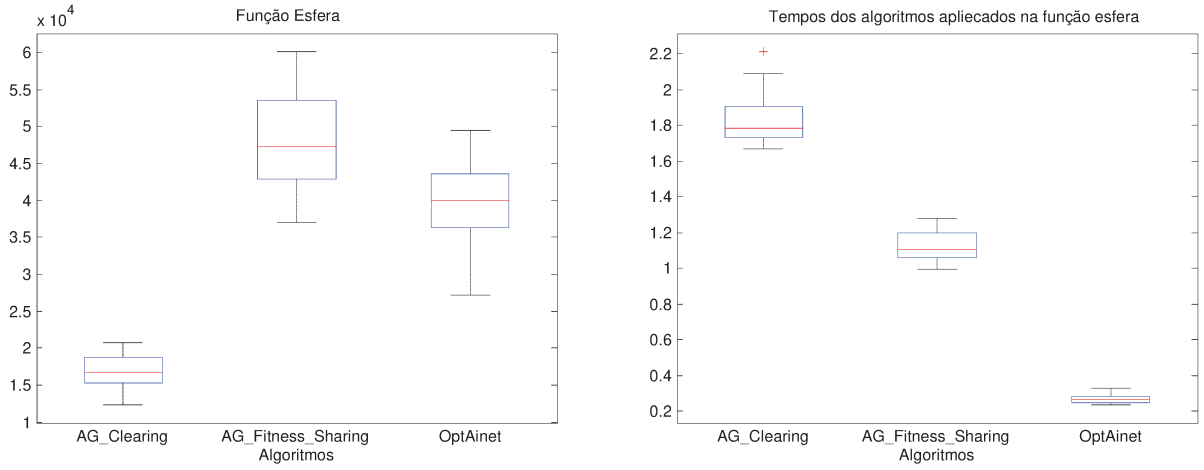
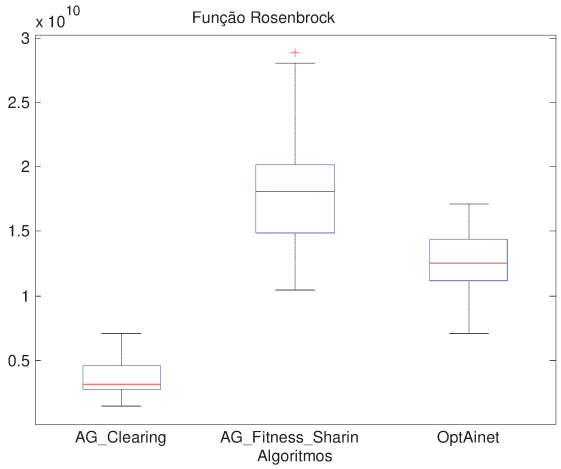


Figura 4.35 a) Boxplot com os resultados dos algoritmos aplicados na função esfera. b) Boxplot com os tempos dos algoritmos aplicados na função esfera.



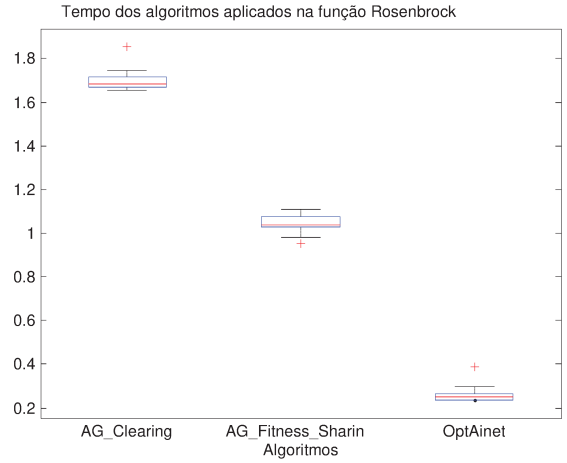
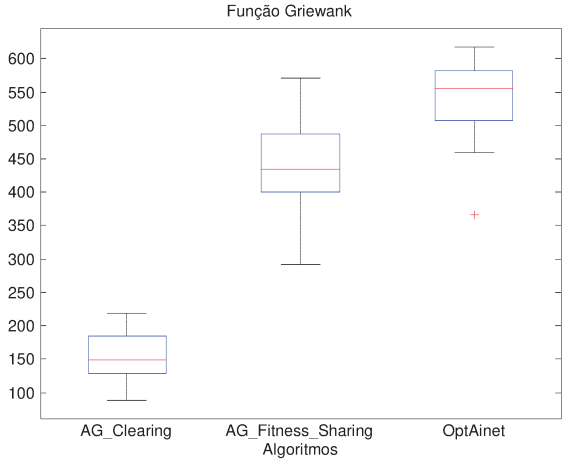


Figura 4.36 a) Boxplot com os resultados dos algoritmos aplicados na função Rosenbrock.

b) Boxplot com os tempos dos algoritmos aplicados na função Rosenbrock.



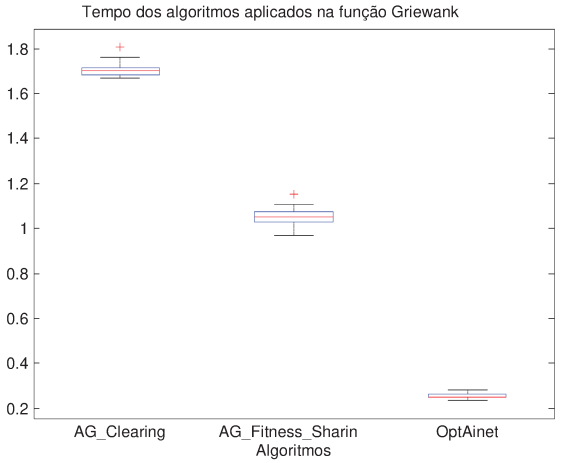
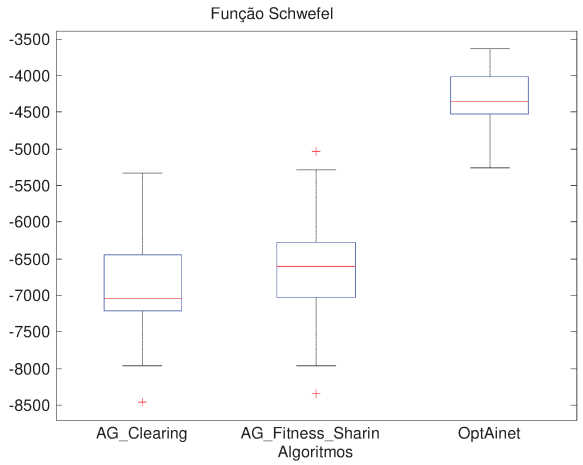


Figura 4.37 a) Boxplot com os resultados dos algoritmos aplicados na função esfera Griewank.

b) Boxplot com os tempos dos algoritmos aplicados na função Griewank.



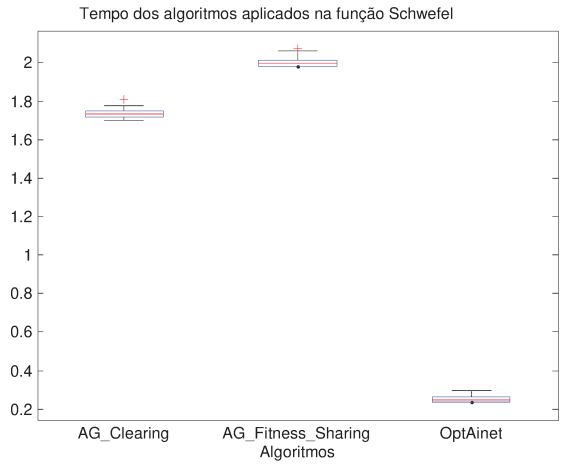
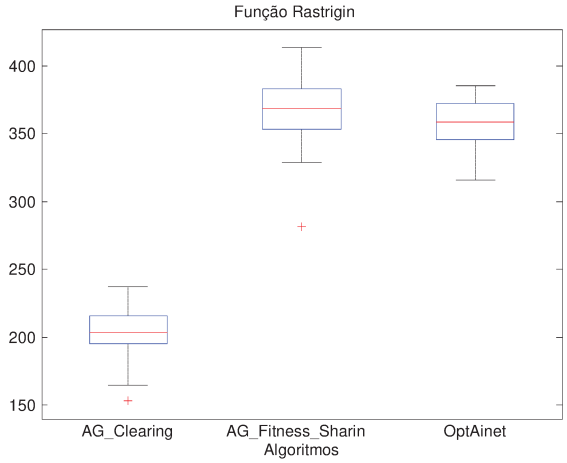


Figura 4.38 a) Boxplot com os resultados dos algoritmos aplicados na função Schwefel.

b) Boxplot com os tempos dos algoritmos aplicados na função Schwefel.



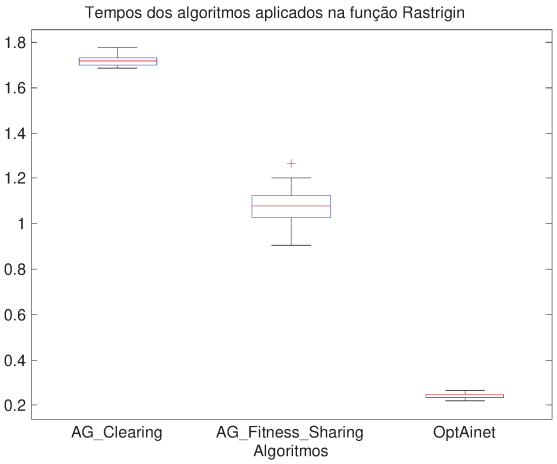
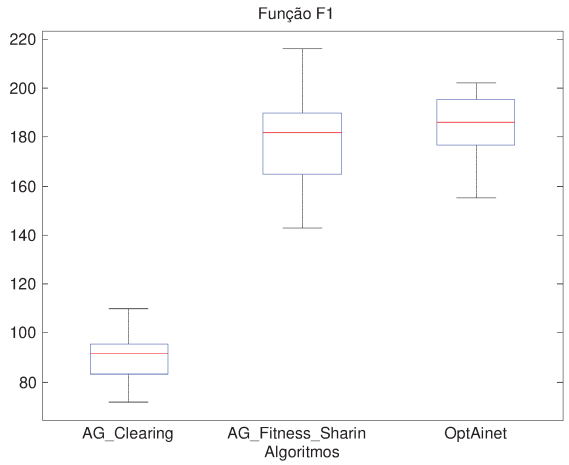


Figura 4.39 a) Boxplot com os resultados dos algoritmos aplicados na função Rastrigin.

b) Boxplot com os tempos dos algoritmos aplicados na função Rastrigin.



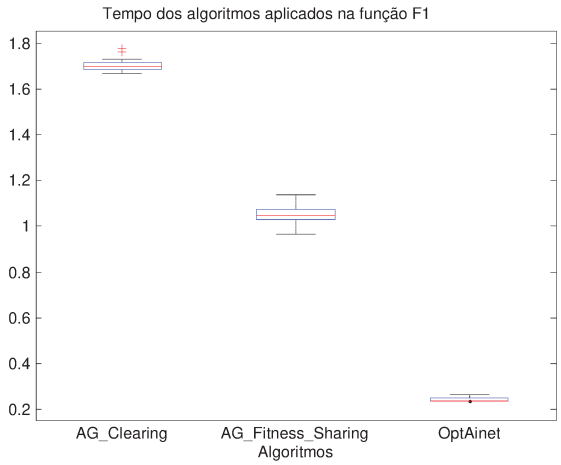
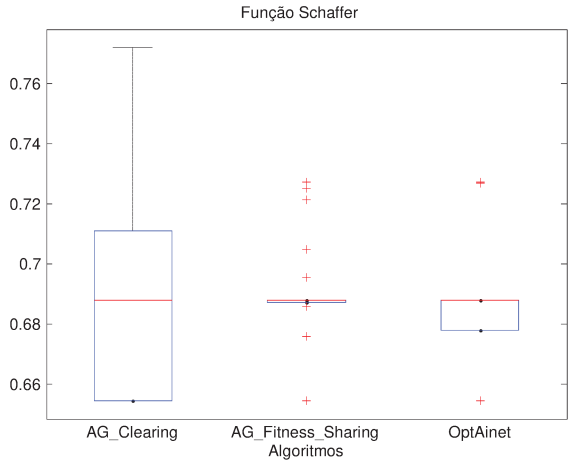


Figura 4.40 a) Boxplot com os resultados dos algoritmos aplicados na função F1.

b) Boxplot com os tempos dos algoritmos aplicados na função F1.



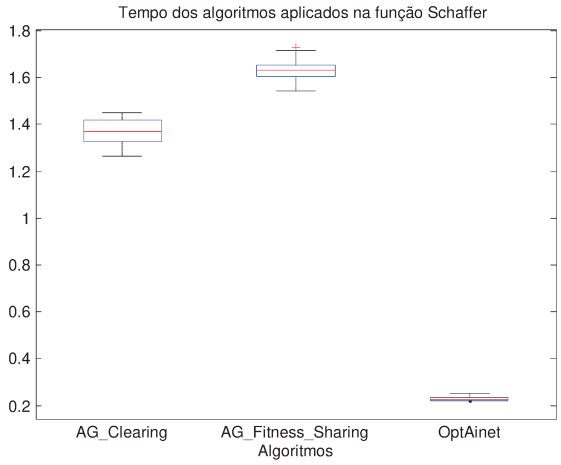


Figura 4.41 a) Boxplot com os resultados dos algoritmos aplicados na função Rastrigin.

b) Boxplot com os tempos dos algoritmos aplicados na função Rastrigin.

Comparando a Tab. 4.1, mais precisamente os algoritmos que utilizaram o *k-means* modificado como pré-processador, com a Tab. 4.5. O melhor resultado da Tab. 4.1 foi melhor em seis das oito funções estudadas. Os melhores resultados da Tab. 4.5 foram obtidos pela aplicação do AG *Clearing* às funções *Schwefel* e F1, nesta última mesmo na Tab. 4.1, o melhor resultado foi obtido pelo CLONALG sem o uso da metodologia proposta.

O algoritmo AG *Clearing* foi melhor do que o AG nas funções esfera, *Griewank*, *Rastringir*, *Schwefel* e F1, entretanto comparado com o KAG, o AG *Clearing* foi melhor apenas nas funções *Schwefel* e F1, ficando na margem do desvio padrão.

De forma geral, a introdução de diversidade obtida pela aplicação da metodologia de busca baseada em clusterização promoveu melhores resultados do que a modificação proposta pelo *Clearing*, que por sua vez foi melhor do que *Fitness Sharing* e o Opt-aiNet.

O melhor resultado obtido pela metodologia de busca baseada em clusterização se deve pelo fato de sua capacidade de encontrar regiões promissoras, e desta forma, posicionar adequadamente os pontos iniciais utilizados pelos algoritmos testados. Outro fator contribuinte é o desperdício de avaliações da função objetivo utilizadas em consequência da manutenção de diversidade ao longo de todas as iterações dos algoritmos de *niching*.

No próximo experimento, o *k-means* modificado será comparado com o *c-means* modificado.

4.4.4 Experimento 4

Com este experimento, tem-se o objetivo de analisar o desempenho dos algoritmos *k-means* modificado e do *c-means* modificado com o mesmo número de avaliações da função objetivo que o *k-means* como pré-processador. Dois critérios de parada foram utilizados, um baseado na estabilidade dos centroides e outro em um número máximo de iterações. A grande diferença em relação aos experimentos anteriores é que os dois algoritmos puderam executar até 2000 iterações para parar, e nos experimentos anteriores o *k-means* como pré-processador dispunha de apenas 20 iterações. Desta forma, havendo convergência dos centroides antes de 20 iterações, ambos estarão em condições de igualdade; não havendo, os resultados apresentados neste experimento serão mais estáveis.

Os principais parâmetros utilizados neste experimento foram:

- Número de centroides = 30;
- Número de dimensões = 30;
- Índice de fuzzificação = 1.25;
- Número de avaliações da função objetivo = 300.

O algoritmo *c-means* modificado apresentou melhores resultados do que o *k-means* em 6 das 7 funções estudadas, e, na função restante, ficou na margem do desvio padrão. Mesmo com os dois algoritmos mantendo o melhor ponto obtido.

Tabela 4.7: Média e desvio padrão calculados com o melhor resultado de cada uma das 30 execuções dos algoritmos – Espaço de busca com 30 dimensões, e 300 avaliações da função objetivo.

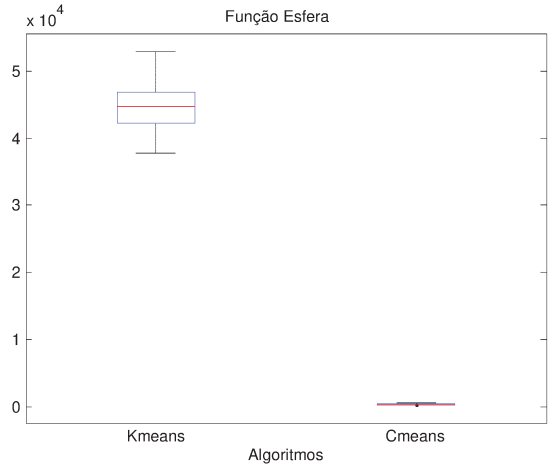
Funções	Algoritmos		
	Global	K-means	C-means
Esfera	0,00	44646±3234	**332±103**
Rosenbrock	0,00	1.47E10±3.45E9	**9.64E5±4.71E5**
Griewank	0,00	411.83±27.67	**4.11±0.78**
Schwefel	-12569,49	-3053±363	**-3064±482**
Rastrigin	0,00	348.17±20.11	**126.23±35.15**
F1	0,00	203.68±10.67	**199.69±10.55**
Schaffer	1,00	**0.70±0.02**	0.67±0.01

O teste t de Student aplicado aos resultados obteve h=0 e p=1 para todos os resultados apresentados.

Tabela 4.8: O intervalo com 95% confiança da média, definido pela aplicação do teste t sobre os resultados dos algoritmos .

Funções	Algoritmos	
	K-means	C-means
Esfera	[43438.63,45853.97]	[293.32,370.67]
Rosenbrock	[1.34E10,1.60E10]	[788432.40,1140198.79]
Griewank	[401.50,422.17]	[3.81,4.40]
Schwefel	[-3189.33,-2917.60]	[-3244.36,-2884.30]
Rastrigin	[340.66,355.68]	[113.11,139.36]
F1	[199.69,207.67]	[195.75,203.64]
Schaffer	[0.69,0.71]	[0.67,0.68]

Os próximos boxplot, apresentados nas Figuras 4.42 até 4.48, contém os resultados obtidos. Além do *c-means* apresentar melhor resultado, obteve-o em menor tempo, pois convergiu mais rapidamente para a estabilidade dos centroides.



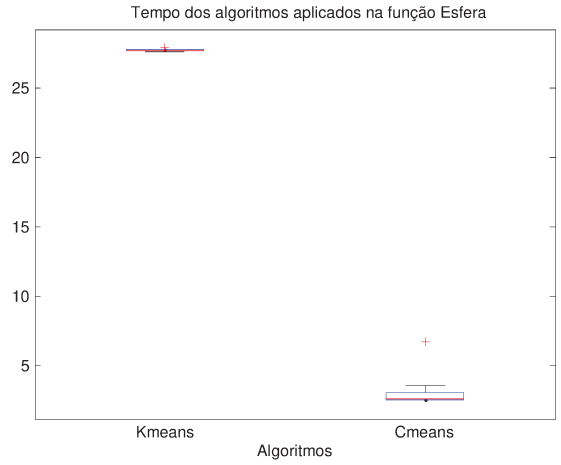
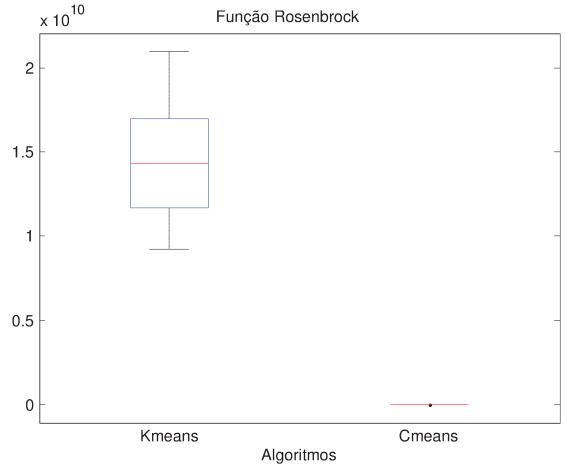


Figura 4.42 a) Boxplot com os resultados dos algoritmos aplicados na função esfera.

b) Boxplot com o tempo dos algoritmos aplicados na função esfera.



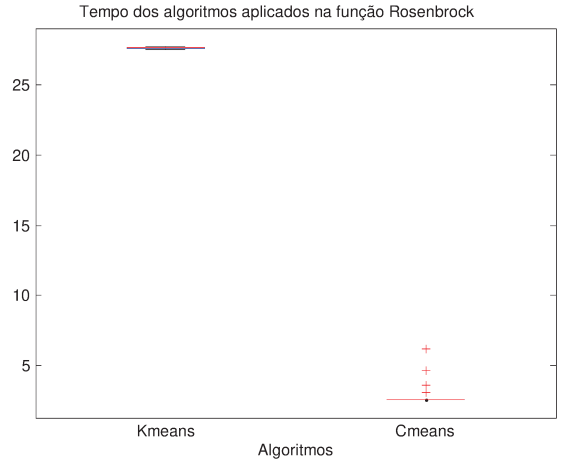


Figura 4.43 a) Boxplot com os resultados dos algoritmos aplicados na função Rosenbrock.

b) Boxplot com o tempo dos algoritmos aplicados na função Rosenbrock.

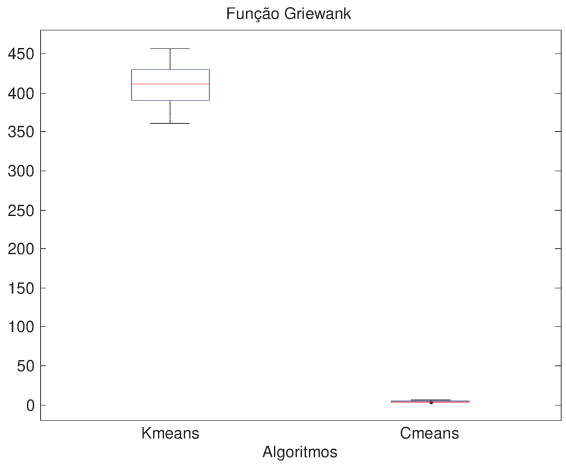
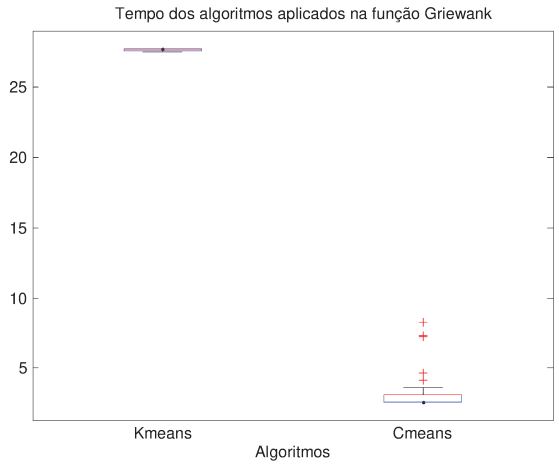


Figura 4.44 a) Boxplot com os resultados dos algoritmos aplicados na função Griewank.



b) Boxplot com o tempo dos algoritmos aplicados na função Griewank.

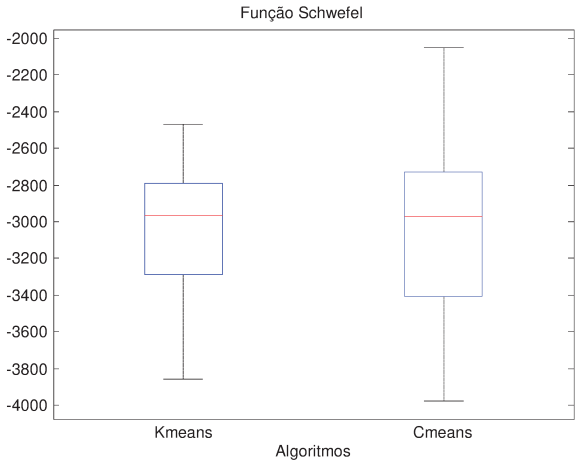
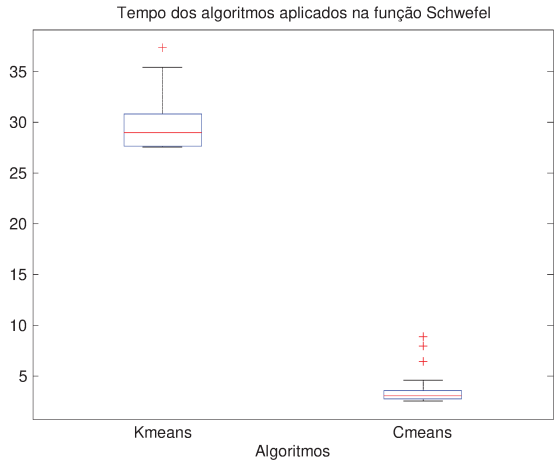
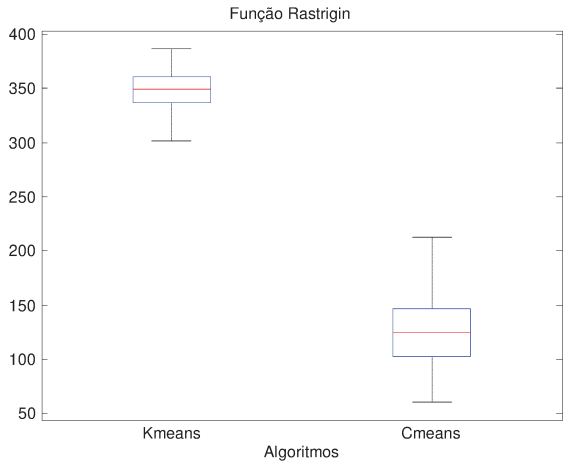


Figura 4.45 a) Boxplot com os resultados dos algoritmos aplicados na função Schwefel.



b) Boxplot com o tempo dos algoritmos aplicados na função Schwefel.



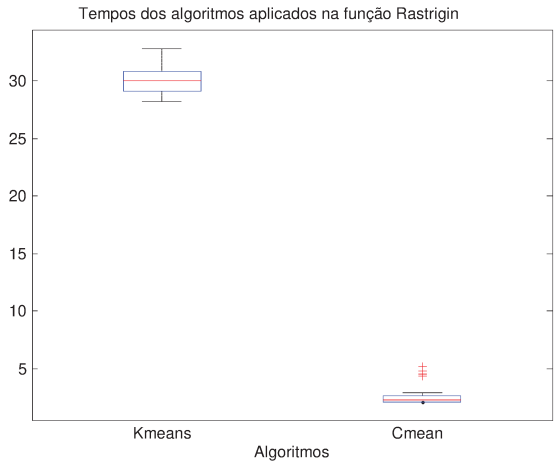
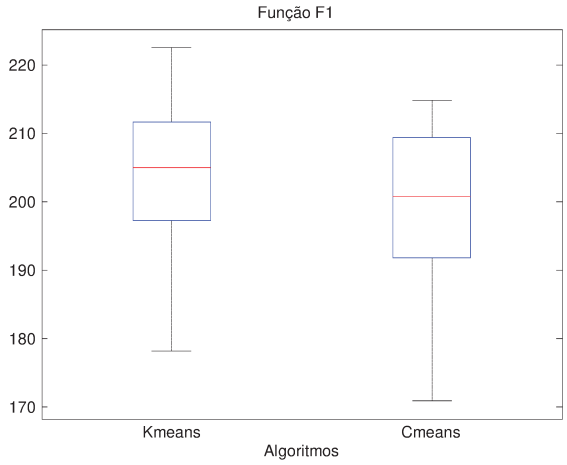


Figura 4.46 a) Boxplot com os resultados dos algoritmos aplicados na função Rastrigin.

b) Boxplot com o tempo dos algoritmos aplicados na função Rastrigin.



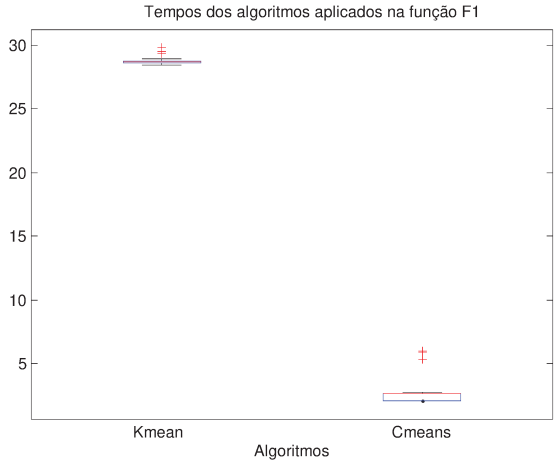
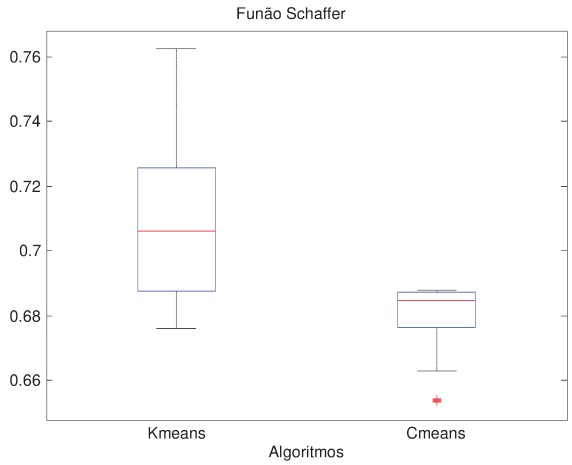


Figura 4.47 a) Boxplot com os resultados dos algoritmos aplicados na função F1.

b) Boxplot com o tempo dos algoritmos aplicados na função F1.



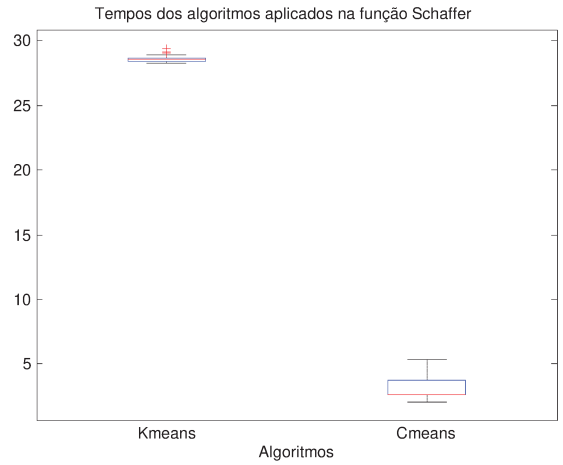


Figura 4.48 a) Boxplot com os resultados dos algoritmos aplicados na função Rastrigin. b) Boxplot com o tempo dos algoritmos aplicados na função Rastrigin.

Neste experimento, o *c-means* modificado apresentou resultados melhores do que o *k-means* modificado. O fato de o *c-means* ser capaz de trabalhar com pertinência nebulosa associada ao peso proposto influenciou positivamente seu desempenho.

Este experimento não caracteriza cabalmente que o *c-means* modificado encontra regiões mais promissoras do que o *k-means* modificado, uma vez que ambos não possuem estratégias de refinamento. Sendo assim, o *k-means* modificado pode posicionar um centroide na base de um ótimo local, enquanto o *c-means* modificado posiciona o centroide no cume de ótimo local. Sendo assim, o *k-means* modificado pode mesmo ter encontrado a região mais promissora, mesmo postando o centroide em um ponto pior avaliado.

Este experimento credencia, não obstante, o *c-means* modificado como um representante bastante adaptado à metodologia proposta na forma em que foi implementada, mas uma análise comparativa mais ampla e detalhada dependerá de esforços a serem realizados em trabalhos futuros.

4.4.5 Experimento 5

Neste experimento, o *k-means* modificado foi utilizado como pré-processador do MPSO, e, desta forma, houve a unificação das duas propostas desta tese (Prampero, Boccato, & Romis, 2011).

Os experimentos foram realizados com cinco funções objetivo, as quais são bem conhecidas na literatura e que já foram discutidas nesta tese. Nos experimentos, duas versões do MPSO foram utilizadas, uma com o *k-means* modificado como pré-processador e outra sem. O MPSO foi configurado com 30 partículas, e avaliou 1.000.000 (um milhão) de vezes cada função objetivo.

O *k-means* modificado como pré-processador utilizou 1000 pontos por partícula para gerar os pontos iniciais do MPSO, o que significa que esta técnica empregou a priori 30.000 avaliações da função objetivo para o pré-processamento. O *k-means* modificado isolado apresentado na Tabela 4.9 gerou 30 centroides utilizando para isto 1.000.000 de pontos distribuídos uniformemente pelo espaço de busca.

Na tabela, o *k-means* modificado isolado mostrou que não foi competitivo em relação ao MPSO, mas o MPSO com *k-means* modificado foi superior ao MPSO sozinho. Este bom posicionamento inicial das partículas favoreceu mesmo um algoritmo que possui estratégias para escapar de ótimos locais.

Tabela 4.9: A média e o desvio padrão do melhor valor encontrado em cada uma das 30 execuções do algoritmo – Espaço de busca com 30 dimensões.

Funções		Algoritmos		
	Global	*K-means modificado*	MPSO com *k-means*	MPSO
Rosenbrock	0,00	49488304.62 ± 8804981.23	**0.03 ± 0.10**	0.17 ± 0.20
Griewank	0,00	291.02 ± 30.81	**0.00 ± 0.00**	**0.00 ± 0.00**
Rastrigin	0,00	300.60 ± 7.01	**0.00 ± 0.00**	**0.00 ± 0.00**
Schaffer_M	30,00	15.86 ± 0.13	**27.30 ± 1.08**	25.05 ± 2.66
Schwefel	-12569,49	-4961.20 ± 153.38	**-12569.41 ± 0.09**	-12569.40 ± 0.12

A tabela anterior mostra que a utilização do *k-means* modificado como pré-processador promove melhoria no desempenho no MPSO, mas não foi capaz de obter bons resultados em sua

execução individual. Isto acontece pois o *k-means* modificado é capaz de encontrar regiões promissoras, mas não consegue refinar as soluções encontradas.

A Figura 4.49 apresenta a convergência do MPSO com e sem o *k-means* modificado.

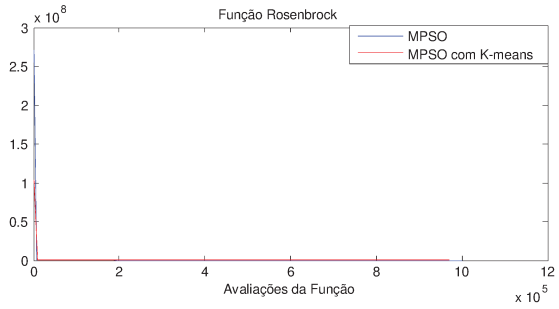
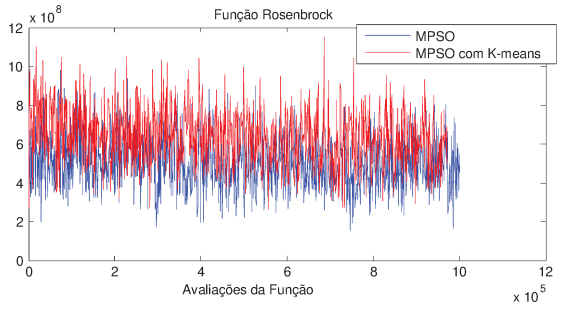


Figura 4.49 a) Convergência da melhor partícula.



b) Convergência da média.

Na Figura 4.49 é visível que a melhor partícula converge rapidamente em ambos os casos, mas o MPSO com *k-means* modificado, apresentou maior diversidade da população, tendo maior amplitude no gráfico de convergência da média. Esta amplitude da média indica variação populacional e agitação, características apresentadas como relevantes no processo de busca multimodal.

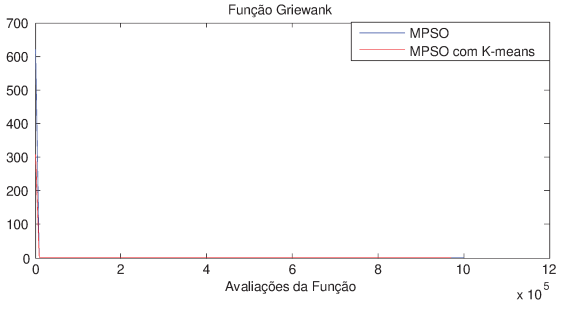
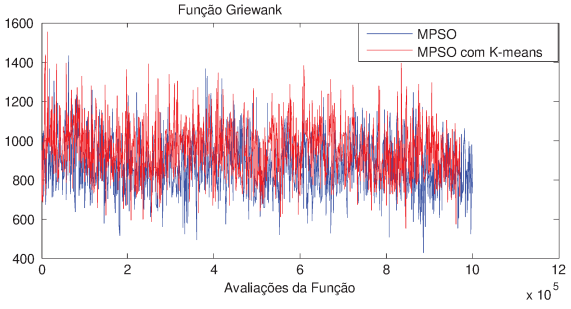
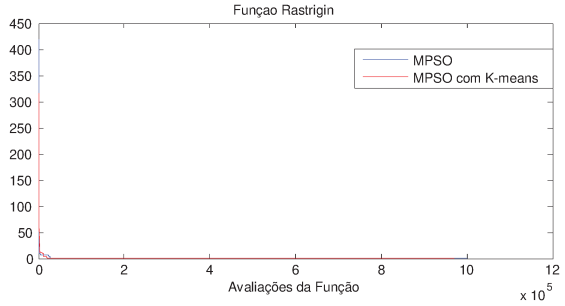


Figura 4.50 a) Convergência da melhor partícula.



b) Convergência da média.

Na Figura 4.51, o *k-means* modificado não promoveu melhorias significativas ao processo, uma vez que os gráficos são similares, tanto para a melhor partícula quanto para a média.



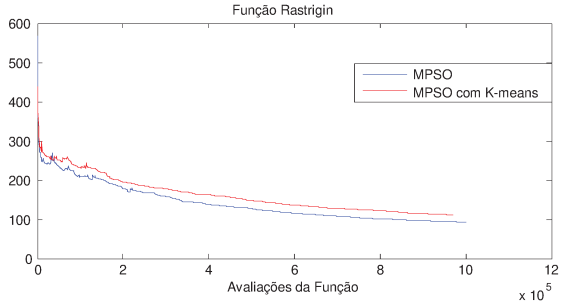
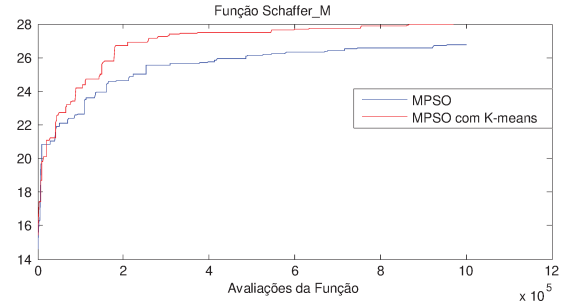


Figura 4.51 a) Convergência da melhor partícula. b) Convergência da media.

Para a função *Schaffer_M*, o uso do *k-means* modificado acelerou o processo devido à melhor condição inicial, como mostra a Fig. 4.52.



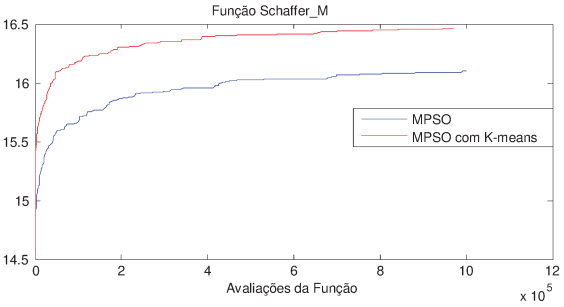
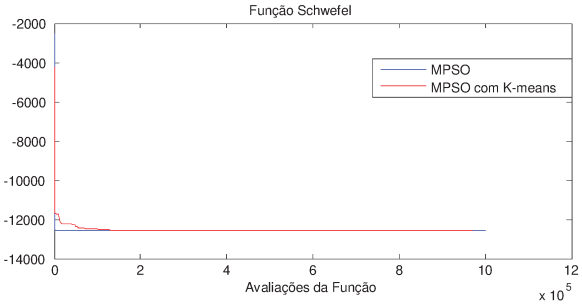


Figura 4.52 a) Convergência da melhor partícula. b) Convergência da média.



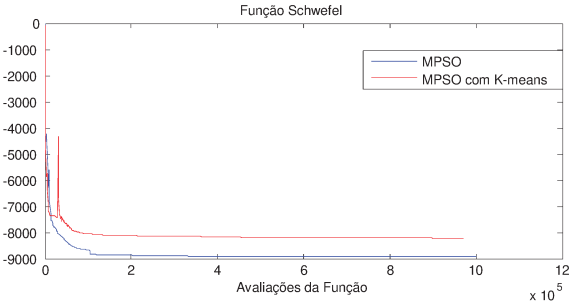


Figura 4.53 a) Convergência da melhor partícula. b) Convergência da média.

A metodologia de busca baseada em clusterização promoveu melhorias, mesmo quando associada com um algoritmo capaz de escapar de ótimos locais e de manter a diversidade. Sendo assim, a proposta detectou regiões promissoras. Neste experimento, para o número utilizado de

avaliações da função objetivo, a posição inicial das partículas influenciou o resultado final do algoritmo.

No próximo teste, o MPSO com e sem *k-means* modificado foram aplicados em um espaço de 1000 dimensões. As demais configurações foram as mesmas, e os resultados estão apresentados na próxima tabela. O *k-means* modificado isolado não foi utilizado, por conta do seu desempenho na simulação anterior.

Tabela 4.10: Média e desvio padrão do melhor valor de cada uma das 30 execuções do algoritmo – Espaço de busca com 1000 dimensões.

Funções		Algoritmos	
	Global	MPSO com *k-means*	MPSO
Rosenbrock	0,00	**0.00 ± 0.00**	2.64 ± 1.59
Griewank	0,00	**0.00 ± 0.00**	**0.00 ± 0.00**
Rastrigin	0,00	0.02 ± 0.02	**0.00 ± 0.00**
Schaffer_M	1000,00	723.52 ± 47.93	**893.08 ± 169.31**
Schwefel	-418982,9	**-418982.88 ± 0.00**	-418955.24 ± 32.55

No experimento com 1000 dimensões, o *k-means* modificado não melhorou o desempenho do MPSO como no contexto de 30 dimensões. Entretanto, provocou melhorias nos resultados de duas funções. A razão para este desempenho é que, os 30.000 pontos utilizados pelo *k-means* modificado para gerar os 30 centroides, tiveram um impacto menos significativo em um espaço de busca com 1000 dimensões, embora os resultados obtidos tenham comprovado a colaboração do *k-means* modificado em alguns casos.

4.5 CONCLUSÕES

Este capítulo apresenta a segunda proposta desta tese, uma metodologia de busca baseada em clusterização, que analisa a função objetivo como uma PDF (do inglês, *Probability Density Function*) e amostra o espaço de busca a partir de pontos espalhados aleatoriamente, estes pontos são utilizados nos algoritmos de clusterização modificados. Tal metodologia foi testada através de uma modificação no algoritmo *k-means*, chamado de *k-means* modificado, que foi utilizado

com um operador de diversidade, um pré-processador que posiciona os pontos em regiões promissoras. Os algoritmos testados utilizaram os pontos gerados pelo *k-means* modificado como população inicial. Uma formulação desse tipo abre a perspectiva de se utilizar todo o arcabouço teórico de metodologias de clusterização como mecanismos auxiliares às ferramentas de busca, bastando para isto um ajuste adequado do peso.

Os ensaios realizados ilustram o potencial do uso de técnicas de clusterização para o processo de busca. O *k-means* modificado mostrou-se eficiente neste quesito, detectando regiões promissoras, e, assim, criando diversidade.

Os resultados mostram que houve uma tendência geral para a melhoria do desempenho em termos de potencial de busca multimodal com a utilização da metodologia baseada em clusterização. Mostram também que a busca pelo ótimo possui dois momentos: uma fase de caráter global, no qual a diversidade é muito importante, e o segundo de caráter local, que necessita da capacidade de refinamento das soluções já encontradas.

Na comparação do *c-means* modificado com o *k-means* modificado, o *c-means* obteve os melhores resultados – essa análise comparativa será aprofundada e ampliada (com a inserção de outras metodologias).

5 CONCLUSÕES E TRABALHOS FUTUROS

Esta tese teve por objetivo geral o desenvolvimento de métodos de otimização para funções multimodais. Para isto, em primeiro lugar, foi desenvolvido o algoritmo MPSO, que tem como principais contribuições o raio de repulsão com controlador de passo, e o armazenamento da direção promissora.

O raio de repulsão tem duas funções básicas: a primeira é a de controlador de passo, que promove movimentação intensa no início do processo de busca, favorecendo a busca global; e movimentação lenta no final, favorecendo a busca local. A segunda é a delimitação de região de atuação, e, com isto, a definição de região de domínio da partícula. Desta forma, cada partícula possui o seu espaço de busca, o que provoca um maior espalhamento das partículas ao longo do domínio factível.

A segunda contribuição foi o armazenamento da direção promissora. Sempre que o MPSO encontra um ponto melhor, armazena a direção associada e continua gerando pontos nesta linha enquanto houver melhora em relação à solução atual. Quando não consegue melhorar, testa as direções perpendiculares, e, ainda não conseguindo, volta a caminhar aleatoriamente.

Com o raio de repulsão, o MPSO apresentou bons resultados nos experimentos realizados, não só em problemas multimodais, mas também em problemas de larga escala. O algoritmo conseguiu, em alguns experimentos, boa convergência da melhor partícula e manutenção da diversidade na população. Ele possui características estruturais relevantes, tais como: é populacional, iterativo, possui controlador de passos, armazena a direção promissora e define a região de atuação da cada partícula. Por conta destas características, apresentou as seguintes propriedades funcionais nos experimentos realizados: rápida convergência da melhor partícula, boa manutenção da diversidade, boa capacidade de refinamento, capacidade de solucionar problemas de larga escala e simplicidade de configuração. Tudo isso foi ilustrado também com a ajuda de um experimento baseado num cenário prático, ligado à otimização de produção de biogás.

A segunda proposta consiste em uma metodologia de busca baseada em clusterização. Esta metodologia mostrou-se eficiente na questão de introduzir diversidade. Cada centroide foi posicionado com base na avaliação dos pontos do seu grupo e na quantidade de pontos do grupo. Desta forma, houve uma tendência de posicionamento em regiões promissoras em termos da presença de ótimos globais.

A metodologia foi implementada na forma de um pré-processador, e o algoritmo *k-means* foi modificado para funcionar de acordo com a proposta. Os resultados do pré-processador comprovaram o acréscimo de diversidade, a um custo baixo em relação ao número de avaliações da função objetivo. Também houve análises com o algoritmo fuzzy c-means, as quais abrem importantes perspectivas de aprofundamento.

Sendo assim, as duas propostas cumpriram seus objetivos e contribuíram, em alguma medida, para a área de otimização. Com o objetivo de que estes estudos continuem, evoluam, e sejam aprimorados; a próxima seção traz alguns possíveis trabalhos futuros.

5.1 TRABALHOS FUTUROS

Nesta seção, serão apresentadas sugestões de trabalhos futuros, algumas já apresentadas durante o texto. Um primeiro esforço sugerido foi a separação do raio de repulsão do algoritmo MPSO, em controlador de passo e controlador de proximidade. O controlador de passo deverá ajustar o tamanho da movimentação da partícula; já o controlador de domínio deverá controlar a proximidade entre as partículas. A sugestão é que o controlador de passo comece grande, e, assim que a partícula melhore sua avaliação, o passo diminui. Dessa forma, a partícula movimenta-se rapidamente no início e lentamente no final, favorecendo o refinamento nesta fase.

O controlador de proximidade poderá começar pequeno, e quando a partícula melhorar, aumentar o raio. Desse modo, quando uma partícula domina um pico de máximo local, o controlador de proximidade inviabiliza a aproximação de outras partículas, fazendo com que estas procurem outros picos.

Outro trabalho importante é o uso do *c-means* e de outros métodos como pré-processadores. Esta aplicação foi sugerida naturalmente pelos resultados do experimento 4.4.5. O *c-means* modificado apresentou melhores resultados do que o *k-means* modificado, este último tendo sido utilizado na maioria dos experimentos sobre otimização baseada em clusterização.

Outra sugestão é o uso de um algoritmo de busca local aplicado aos pontos encontrados pelo *c-means* como pré-processador. Desta forma, o algoritmo de busca local deverá refinar as soluções encontradas pelo *c-means*. Espera-se que os resultados sejam competitivos, uma vez que

o *c-means* encontrou regiões promissoras e posicionou pontos nestas regiões, mesmo que ainda na base de alguns picos.

REFERÊNCIAS BIBLIOGRÁFICAS

Afshar, M. H., Ketabchi, H., & Rasa, E. (2006). Elitist continuous ant colony optimization algorithm: application to reservoir operation problems. *International Journal of Civil Engineerng, 4*(4).

Bäck, T., Fogel, D. B., & Michalewicz, Z. (2000). *Evolutionary computation 1: basic algorithms and operators.* Institute of Physics Publishing.

Bäck, T., Fogel, D. B., & Michalewicz, Z. (2000b). *Evolutionary computation 2: advanced algorithms and operators.* Institute of Physics Publishing.

Bai, Q. (2010). Analysis of particle swarm optization algorithm. *Computer and Information Science*, 180-184.

Baluja, S. (1994). *Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning.* Pittsburgh: Carnegie Mellon University.

Bazaraa, M. S., Shetty, C. M., & Sherali, H. D. (1993). *Nonlinear programming: theory and algorithms* (2 ed.). New York: John Wiley & Sons.

Bergh, F. v., & Engelbrecht, A. P. (2004). A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation, 8*, 225-239.

Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms.* New York: Plenum Press.

Birbil, S. I., & Fang, S. C. (2003). An electromagnetism-like mechanism for glocal optimization. *Journal of Global Optimization, 25*, 263-283.

Birbil, S. I., Fang, S. C., & Sheu, L. R. (2004). On convergence of a population-based global optimization algorithm. *Journal of Global Optimization, 30*, 301-318.

Blackwell, T., & Bentley, P. (2002). Dynamic Search with Charged Swarms. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 19-26.

Brandão, M. A. (2010). *Estudo de alguns métodos determinísiticos de otimização irrestrita.* Uberlândia: Dissertação (Mestrado) Universidade Federal de Uberlândia.

Brown Jr, G. H. (1977). On Halley's variation of Newton's method. *American Mathematical Monthly, 84*(9), 726-728.

Brufati, T. E. (2011). *Método de gradientes conjugados.* Curitiba: Departamento de Matemática da Universidade Federal do Paraná.

Camponogara, E. (09 de 2006). *Métodos de otimização: teoria e prática.* Acesso em 30 de 04 de 2013, disponível em http://www.das.ufsc.br/~camponog/Disciplinas/DAS-9011/LN.pdf

Carvalho, D. R., & Freitas, A. A. (2002). Uma revisão de métodos de niching para algoritmos genéticos. *29*, 97-118. Acesso em 28 de março de 2013, disponível em http://www.utp.br/tuiuticienciaecultura/FACET/FACET%2029/PDF/Art%207.pdf

Castro, L. N. (2006). *Fundamentals of natural computing: basic concepts, algorithms, and aplication.* Flórida: Chapman & Hall / CRC Press.

Castro, L. N., & Timmis, J. (2002). An artificial immune network for multimodal function optimization. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'02)*, *1*, pp. 674-699. May, Hawaii.

Castro, L. N., & Timmis, J. (2002b). *Artificial immune systems: a new computational intelligence approach.* Springer Verlag.

Castro, L. N., & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems, 3*, 239-251.

Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Application, 45*, pp. 41-45.

Chen, B. R., & Feng, X. T. (2005). Particle swarm optimization with contracted ranges of both search space and velocity. *Journal of Northeastern University: Natural Science, 26*(5), 488-491.

Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *Proceedings of the Congress on Evolutionary Computation*, pp. 1951-1957.

Coelho, G. P. (2011). *Redes imunológicas artificiais para otimização em espaços contínuos: uma proposta baseada em concentração de anticorpos.* Campinas: Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

Costa Junior, N., Prampero, P. S., Assad, M. M., & Attux, R. R. (2011). Estimate of biogas generation in landfills using natural computation techniques. *Brazilian Congress of Mechanical Engineering (COBEM).* Natal.

De Bonet, J. S., Isbell, C. L., & Viola, P. (1997). MIMIC: finding optima by estimating probability densities. (J. M. Mozer MC, Ed.) *In Advances in Neural Information Processing Systems, 9*, pp. 424-430.

Deb, K. (2004). *Multi-objective optimization using evolutionary algorithms.* England: John Wiley & Sons Ltd.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification.* John Wiley & Sons.

Dunn, J. C. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 32-57.

Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the Congress on Evolutionary Computating*, pp. 84-88.

Eberhart, R., & Shi, Y. (2001). Particle swarm optimization: developments, aplications and resources. *Proceedings of the 2001 Congress on Evolutionary Computation.* Seoul.

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction* (2 ed.). Chichester, England: Jonh Wiley and Sons.

Ensinas, A. V. (2003). *Estudo da geração de biogás no aterro sanitário Delta em Campinas SP.* Campinas: UNICAMP, Faculdade de Engenharia Mecânica .

Glover, F. W., & Laguna, M. (2004). Tabu Search. *Springer.*

Glover, F., & Marti, R. (2006). Tabu search. *Metaheuristic Procedures for Training Neural Networks*, 53-69.

Goldbarg, M. C., & Luna, H. P. (2005). *Otimização combinatória e programação linear: modelos e algoritmos.* Rio de Janeiro: Campus.

Gomes, L. C. (2006). *Inteligência computacional na síntese de meta-heurísticas para otimização combinatória e multimodal.* Campinas: Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

Gonçalves, A. R. (2011). *Otimização em ambientes dinâmicos com variáveis.* Campinas: Dissertação (mestrado) Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* University of Michigan Press.

Honarkhah, M., & Caers, J. (2010). Stochastic simulation of patterns using distance-based pattern modeling. 487-517.

Hopcroft, J. E., Ullman, J. D., & Motwani, R. (2002). *Introdução à teoria de autômatos, linguagens e computação.* Rio de Janeiro: Campus.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 264-323.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys, 31*, pp. 264-323.

Kennedy , J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, (pp. 1942-1948).

Kennedy, J., Eberhart , R. C., & Shi, Y. (2001). *Swarm intelligence.* San Francisco: Morgan Kaufmann / Academic Press.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, pp. 671- 680.

Kohonen, T. (1995). *Self-organizing maps.* Berlin: Springer.

Kraus, J. D., & Fleisch, D. A. (1999). *Electromagnetics with applications* (5 ed.). McGraw-Hill.

Krishnanand, K., & Ghose, D. (2009). Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence, 3*, 87-124.

Larrañaga, P., & Lozano, J. A. (2002). Estimation of distribution algorithms: a new tool for evolutionary computation. *Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers*.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 129-137.

Lopes, H. S. (2006). *Fundamentos da computação evolucionária e aplicações.* (Vol. 8). Bandeirantes: Escola Regional de Informática da SBC.

Lu, Z. S., & Hou, Z. R. (2004). Particle swarm optimization with adaptive mutation. *Acta Electronica Sinica, 32*(3), 416-420.

Luenberger, D. G. (1984). *Linear and nonlinear programming* (2ª ed.). Califórnia: Addison-Wesley.

Luenberger, D. G., & Ye, Y. (2008). *Linear and Nonlinear Programming.* Springer.

Madeiro, S. S. (2011). Density as the segregation mechanism in fish school search for multimodal optimization problems. *Proceedings of the Second International Conference on Advances in Swarm Intelligence*, pp. 563-572.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms.* University of Illinois at Urbana-Champaign.

Masehian, E., & Sedighizadeh, D. (2009, December). Particle swarm optimization: methods, taxonomy and applications. *International Journal of Computer Theory and Engineering, 1*, 486-502.

McQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 281-297.

Michalewicz, Z. (1996). *Genetics algorithms + data structures = evolution programs* (3 ed.). New York: Springer-Verlag Berlin Hildelberg.

Michalewicz, Z., & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation, 4*(1), 1-32.

Molga, M., & Smutnicki, C. (2005). *Test functions for optimization needs*. Retrieved 01 18, 2013, from http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf

Monson, C. K., & Seppi, K. D. (2004). The Kalman swarm: a new approach to particlemotion in swarm optimization. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 140-150.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV*, pp. 178-187.

Mühlenbein, H., Mahnig, T., & Ochoa , A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 213-247.

Mühlenbein, H., Mahnig, T., & Ochoa, A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics, 5*, 213-247.

Oliveira, F. R. (2012). *Estudos de alguns métodos classicos de otrimização restrita não-linear.* Uberlândia: Universidade Federal de Uberlândia - Faculdade de Matemática.

Passaro, A., & Sarita, A. (2006). Clustering particles for multimodal function optimization. *In: Proc. GSICE/WIVA. published on CD, ISSN 1970-5077.*

Pedrycz, W., & Gomide, F. (2007). *Fuzzy systems engineering toward human-centric computing.* Wiley.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, (pp. 5-20).

Pelikan, M., Goldberg, D., & Cantú-Paz, E. (2000). BOA: the bayesian optimization algorithm. *Evolutionary Compututation*, 311-340.

Petrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. *IEEE 3rd International Conference on Evolutionary Computation*, (pp. 798–803). Nagoya.

Prampero, P. S., & Attux, R. (2011). Magnetic particle swarm optimization with estimation of distribution. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2011).* New Orleans, USA.

Prampero, P. S., & Attux, R. (2011a). Magnetic particle swarm optimization. *Proceedings of the IEEE Symposium on Swarm Intelligence.* Paris.

Prampero, P. S., & ATTUX, R. R. (2013). Magnetic particle swarm optimization. *Journal of Artificial Intelligence and Soft Computing Research, 2*, 5.

Prampero, P. S., Boccato, L., & Romis, A. (2011). Multimodal optimization as a clustering task: general formulation and application in the context of particle swarm. *In: 10th Brazilian Congress on Computational Intelligence (CBIC 2011).* Fortaleza: Brazilian Society on Computational Intelligence (SBIC).

Price, K. (1997). Differential evolution vs. the functions of the 2nd ICEO. *In Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, (pp. 153-157). Indianapolis, USA.

Price, K. V., Storn, R. M., & Lampinen, J. A. (2005). *Differential evolution: a practical approach to global optimization.* Springer-Verlag.

Russell, S. J., & Norvig, P. (2004). *Inteligência artificial.* Rio de Janeiro: Campus.

Sabin, M., & Gray, R. (1986). Global convergence and empirical consistency of the generalized Lloyd algorithm. *IEEE Transactions on Information Theory*, 148-155.

Sedighizadeh, D., & Masehian, E. (2009). Particle swarm optimization methods, taxonomy and applications. *International Journal of Computer Theory and Engineering, 1*(5), 1893-1901.

Serapião, A. B. (Agosto de 2009). Fundamentos de otimização por inteligência de enxames: uma visão geral. *Sociedade Brasileira de Automática: Controle & Automação, 20*(3), 271-304.

Shi, Y., & Eberhart, R. C. (1998). A modified particle swam optimizer. *IEEE Word Congress on Computational Intelligence*, pp. 69-73.

Silva, E. C. (20 de 10 de 2013). *Otimização aplicada ao projeto se sistemas mecânicos* . Acesso em 20 de 10 de 2013, disponível em Escola Politécnica: Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos: http://sites.poli.usp.br/d/pmr5215/notas.html

Storn, R., & Price, K. (1995). Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces. *International Computer Science Institute.* Berkeley.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., & Tiwari, S. (2005). *Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization.* Retrieved from www.lri.fr/~hansen/Tech-Report-May-30-05.pdf

Sun, J., Zhang , Q., & Tsang, E. P. (2005). DE/EDA: a new evolutionary algorithm for global optimization. *Information Sciences*.

Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., & Yang, Z. (2008). *Benchmark functions for the CEC'2008 special session and competition on large scale global optimization.* Retrieved from http://sci2s.ugr.es/programacion/workshop/Tech.Report.CEC2008.LSGO.pdf

Tillet, J., Rao, T. M., Sahin, F., & Rao, R. (2005). Darwian particle swarm optimization. *Proceedings of the 2nd Indian International Conference on Artificial Intelligence*, (pp. 1474-1487).

US EPA. (2013, 07 19). *Landfill Gas Emissions Model (LandGEM)*. Retrieved from United States Environmental Protection Agency: http://www.epa.gov/ttncatc1/products.html#software

Von Zuben, F. J. (1996). *Modelos paramétricos e não-paramétricos de redes neurais artificiais e aplicações.* Tese de Doutorado, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica , Campinas.

White, T., & Pagurek, B. (1998). Towards multi-swarm problem solving in networks. *Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98)*, (pp. 333-340).

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67-82.

Zhang, T. (2006). *Filter-based training pattern classification for spatial pattern simulation.* PhD thesis, Stanford University.

Zhang, W., & Xie, X. (2003). DPSO: Hybrid particle swarm with differential evolution operator. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 3816-3821.

APÊNDICE A – Funções Utilizadas

Segue abaixo as funções utilizadas nos experimentos realizados. Tais funções foram escolhidas por estarem presentes nos *benchmarks* (Suganthan, et al., 2005) e (Tang, et al., 2008) de otimização e comporem um conjunto satisfatório de testes. Seguem as funções:

1) Função Esfera

$$F(x) = \sum_{i=1}^{n} x_i^2 \tag{A.1}$$

onde $\mathbf{x} \in [-100,100]^n$. O ponto de mínimo global é $\mathbf{x}^*=[0,0,\ldots,0]$.

2) Função de Rosenbrock

$$F(x) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \tag{A.2}$$

onde $\mathbf{x} \in [-30,30]^n$ e o ponto de mínimo global é $\mathbf{x}^*=[1,1,\ldots,1]$.

3) Função Griewank

$$F(x) = 1 + 0.00025 \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}} \tag{A.3}$$

onde $\mathbf{x} \in [-600,600]^n$. O ponto de mínimo global é $\mathbf{x}^*=[0,0,\ldots,0]$.

4) Função Rastrigin

$$F(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)] \tag{A.4}$$

onde $\mathbf{x} \in [-5.12, 5.12]^n$ e o ponto de mínimo global é $\mathbf{x}^*=[0,0,\ldots,0]$.

5) Função Salomon

$$F(x) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2} \quad \text{(A.5)}$$

onde $\mathbf{x} \in [-100,100]^n$ e o ponto de mínimo global é $\mathbf{x}^*=[0,0,\ldots,0]$.

6) Função Schwefel

$$F(x) = -\sum_{i=1}^{n} x_i \sin\sqrt{|x_i|} \quad \text{(A.6)}$$

onde $\mathbf{x} \in [-500,500]^n$ e O ponto de mínimo global é $\mathbf{x}^*=[420.97,\ldots,420.97]$.

7) Função Levy

$$F(x) = sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i - 1)^2\left(1 + 10sin^2(\pi y_i + 1)\right)\right] + (y_n - 1)^2\left(10sin^2(2\pi y_n)\right) \quad \text{(A.7)}$$

onde $y_i = 1 + \frac{(x_i - 1)}{4}$

, $\mathbf{x} \in [-10,10]^n$ e o ponto de mínimo global é $\mathbf{x}^*=[1,1,\ldots,1]$.

8) Função F1

$$F(x) = \sum_{i=1}^{n} |x_i - i| \quad \text{(A.8)}$$

,onde $\mathbf{x} \in [-100, n+1]^n$ e o ponto de mínimo global é $\mathbf{x}^*=[1,2,\ldots,n]$.

A função F1 não está presente nos *benchmark*s citados, mas foi incluída nos estudos por possuir características ruins para a aplicação das técnicas desenvolvidas. Esta função é estritamente unimodal e não possui mínimos locais.

9) Schaffer

$$F(x) = 0.5 - \left(\frac{\sin\left(\sqrt{\sum_{i=1}^{n}(x_i)^2}\right)^2 - 0.5}{\left(0.001\left(\sum_{i=1}^{n}(x_i)^2\right)+1\right)^2} \right) \tag{A.9}$$

onde $\mathbf{x} \in [-10,10]^n$ e o ponto de máximo global é $\mathbf{x}^*=[0,0,\ldots,n]$.

10) Schaffer_M

$$F(x) = \sum_{i=1}^{n-1} 0.5 + \left(\frac{\sin\left(\sqrt{(x_{i+1})^2+(x_i)^2}\right)^2 - 0.5}{\left(0.001\left((x_{i+1})^2+(x_i)^2\right)+1\right)} \right) \tag{A.10}$$

onde $\mathbf{x} \in [-100,100]^n$ e o ponto de mínimo global é $\mathbf{x}^*=[0,0,\ldots,n]$.