UNIVERSIDADE FEDERAL DE GOIÁS
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA
E DE COMPUTAÇÃO

COMPARAÇÃO DE ALGORITMOS DE ENXAME DE PARTÍCULAS PARA OTIMIZAÇÃO DE PROBLEMAS EM LARGA ESCALA

Leonardo Alves Moreira de Melo

[UFG] & [EMC]
[Goiânia - Goiás - Brasil]
27 de novembro de 2018

sistema de bibliotecas ufg

PRPG
PRÓ-REITORIA DE PÓS-GRADUAÇÃO

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação:

Nome completo do autor: Leonardo Alves Moreira de Melo

Título do trabalho: Comparação de algoritmos de enxame de partículas para otimização de problemas em larga escala

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Leonardo Alves M. de Melo

Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Assinatura do(a) orientador(a)[2]

Data: 28 / 11 / 18

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

[2] A assinatura deve ser escaneada.

UNIVERSIDADE FEDERAL DE GOIÁS
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA
E DE COMPUTAÇÃO

COMPARAÇÃO DE ALGORITMOS DE ENXAME DE PARTÍCULAS PARA OTIMIZAÇÃO DE PROBLEMAS EM LARGA ESCALA

Leonardo Alves Moreira de Melo

Dissertação apresentada à Banca Examinadora como exigência parcial para a obtenção do título de Mestre em Engenharia Elétrica e de Computação pela Universidade Federal de Goiás (UFG), Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), sob a orientação do Prof. Dr. Gelson da Cruz Júnior

[UFG] & [EMC]
[Goiânia - Goiás - Brasil]
27 de novembro de 2018

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Alves Moreira de Melo, Leonardo
COMPARAÇÃO DE ALGORITMOS DE ENXAME DE PARTÍCULAS PARA OTIMIZAÇÃO DE PROBLEMAS EM LARGA ESCALA [manuscrito] / Leonardo Alves Moreira de Melo. - 2018.
lxiv, 64 f.: il.

Orientador: Prof. Dr. Gelson da Cruz Junior.
Dissertação (Mestrado) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Goiânia, 2018.
Bibliografia.
Inclui siglas, abreviaturas, lista de figuras, lista de tabelas.

1. Enxame de Partículas. 2. PSO. 3. Otimização. I. da Cruz Junior, Gelson, orient. II. Título.

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO

Ata de Dissertação de Mestrado

Ata da sessão de julgamento da Dissertação de Mestrado em Engenharia Elétrica e de Computação, área de concentração Engenharia de Computação, do candidato **Leonardo Alves Moreira de Melo,** realizada em 28 de setembro de 2018.

Aos vinte e oito dias do mês de setembro de dois mil e dezoito, às 10:00 horas, na na Sala 104 do Centro de Aulas da Escola de Engenharia Elétrica e de Computação (EMC), Universidade Federal de Goiás (UFG), reuniram-se os seguintes membros da Comissão Examinadora designada pela Coordenadoria do Programa de Pós-graduação em Engenharia Elétrica e de Computação: Os Doutores , Gelson da Cruz Junior – Orientador (EMC/UFG), Cássio Leonardo Rodrigues – INF/UFG e Karina Rocha Gomes da Silva – EMC/UFG, para julgar a Dissertação de Mestrado de **Leonardo Alves de Melo,** intitulada **"COMPARAÇÃO DE ALGORITMOS DE ENXAME DE PARTÍCULAS PARA OTIMIZAÇÃO DE PROBLEMAS EM LARGA ESCALA"**, apresentada pelo Candidato como parte dos requisitos necessários à obtenção do grau de Mestre, em conformidade com a regulamentação em vigor. O Professor Doutor Gelson da Cruz Junior da Comissão, abriu a sessão e apresentou o candidato que discorreu sobre seu trabalho, após o que, foi arguido pelos membros da Comissão na seguinte ordem: Cássio Leonardo Rodrigues e Karina Rocha Gomes da Silva. A parte pública da sessão foi então encerrada e a Comissão Examinadora reuniu-se em sessão reservada para deliberar. A Comissão julgou então que o Candidato, tendo demonstrado conhecimento suficiente, capacidade de sistematização e argumentação sobre o tema de sua Dissertação, foi considerado **aprovado** e deve satisfazer as exigências listadas na Folha de Modificação de Dissertação de Mestrado, em anexo a esta Ata, no prazo máximo de 60 dias, ficando o professor orientador responsável por atestar o cumprimento dessas exigências. Os membros da Comissão Examinadora descreveram as justificativas para tal avaliação em suas respectivas Folhas de Avaliação, anexas a esta Ata. Nada mais havendo a tratar, o presidente da Comissão declarou encerrada a sessão. Nos termos do Regulamento Geral dos Cursos de Pós-graduação desta Universidade, a presente Ata foi lavrada, lida e, julgada conforme, segue assinada pelos membros da Comissão supracitados e pelo Candidato. Goiânia, 28 de setembro de 2018.

Comissão Examinadora Designada:

Gelson da Cruz Junior – Orientador (EMC/UFG) (Avaliação: APROVADO)

Cássio Leonardo Rodrigues – INF/UFG (Avaliação: APROVADO)

Karina Rocha Gomes da Silva – EMC/UFG (Avaliação: APROVADO)

Candidato:

Leonardo Alves Moreira de Melo

"O único motivo por que as pessoas não têm o que desejam é que elas pensam mais no que não desejam do que naquilo que de fato desejam".

John Assaraf
em *"The Secret"*, November 26, 2006.

A todas as pessoas que acreditaram em mim.

AGRADECIMENTOS

Gostaria de agradecer a oportunidade que a Universidade Federal de Goiás (UFG) e a Escola de Engennharia Elétrica, Mecânica e de Computação (EMC) me deram de poder continuar meus estudos acadêmicos. Em especial ao Gelson da Cruz Júnior que além de meu professor e orientador cada vez mais me acompanhou nessa jornada como um amigo. Também agradeço aos amigos Kayo Fernandes Pimentel e Matheus Ullmann pelo companheirismo no estudo e produção deste trabalho.

RESUMO

O número de algoritmos baseados na otimização por enxame de partículas (PSO) aplicados para resolver problemas de otimização em grande escala (até 2.000 variáveis) aumentou significativamente. Este trabalho apresenta análises e comparações entre cinco algoritmos (CCPSO2, LSSPSO, OBL-CPSO, SPSO e VCPSO). Testes foram realizados para ilustrar a eficiência e viabilidade de usar os algoritmos para resolver problemas em larga escala. Seis funções de referência que são comumente utilizadas na literatura (Ackley 1, Griewank, Rastrigin, Rosenbrock, Schwefel 1.2 e Sphere) foram utilizadas para testar a performance desses algoritmos. Os experimentos foram realizados utilizando um problema de alta dimensionalidade (500 variáveis), variando o número de partículas (50, 100 e 200 partículas) em cada algoritmo, aumentando assim a complexidade computacional. A análise mostrou que os algoritmos CCPSO2 e OBL-CPSO mostraram-se significativamente melhores que os outros algoritmos para problemas multimodais mais complexos (que mais se assemelham a problemas reais). No entanto, considerando as funções unimodais, o algoritmo CCPSO2 destacou-se perante os demais. Nossos resultados e análises experimentais sugerem que o CCPSO2 e o OBL-CPSO são algoritmos de otimização altamente competitivos para resolver problemas de otimização complexos e multimodais em larga escala.

COMPARISON OF PARTICLE SWARM OPTIMIZATION ALGORITHMS FOR LARGE SCALE PROBLEMS

ABSTRACT

In order to address an issue concerning the increasing number of algorithms based on particle swarm optimization (PSO) applied to solve large-scale optimization problems (up to 2000 variables), this article presents analysis and comparisons among five state-of-the-art PSO algorithms (CCPSO2, LSS-PSO, OBL-PSO, SPSO and VCPSO). Tests were performed to illustrate the efficiency and feasibility of using the algorithms for this type of problem. Six benchmark functions most commonly used in the literature (Ackley 1, Griewank, Rastrigin, Rosenbrock, Schwefel 1.2 and Sphere) were tested. The experiments were performed using a high-dimensional problem (500 variables), varying the number of particles (50, 100 and 200 particles) in each algorithm, thus increasing the computational complexity. The analysis showed that the CCPSO2 and OBL-PSO algorithms found significantly better solutions than the other algorithms for more complex multimodal problems (which most resemble realworld problems). However, considering unimodal functions, the CCPSO2 algorithm stood out before the others. Our results and experimental analysis suggest that CCPSO2 and OBL-PSO seem to be highly competitive optimization algorithms to solve complex and multimodal optimization problems.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

CCPSO2	–	*New Cooperative Coevolving Particle Swarm Optimization*
LSSPSO	–	*Particle Swarm Optimization using Local Stochastic Search and Enhancing Diversity*
OBL-CPSO	–	*Opposition-Based Learning Competitive Particle Swarm Optimizer*
PNL	–	Programação Não Linear
PSO	–	*Particle Swarm Optimization*
SIN	–	Sistema Integrado Nacional
SPSO	–	*Standard Particle Swarm Optimization*
VCPSO	–	*Vector Coevolving Particle Swarm Optimization Algorithm*

CAPÍTULO 1

INTRODUÇÃO

Com o passar do tempo uma grande quantidade de problemas complexos surgiram no mundo, esses problemas se caracterizam pela dificuldade de se resolver. Uma alternativa muito válida para a resolução de problemas em larga escala é a utilização de algoritmos e metaherísticas. Com o poder computacional, problemas que precisavam muitas vezes de meses para serem resolvidos agora são resolvidos em questão de horas. Um tipo de algoritmo específico se destaca nessa função, os algoritmos baseados em otimização por enxame de partículas. Desde o desenvolvimento do primeiro algoritmo de enxame de partículas (*Particle Swarm Optimization* - PSO) (KENNEDY; EBERHART, 1995) vários algoritmos baseados em enxame de partículas foram criados. O objetivo principal da maioria desses algoritmos é a resolução de problemas em pequena escala, ou seja, com poucas variáveis e baixa complexidade.

Uma vertente do estudo de metaherísticas busca resolver problemas mais complexos com grande quantidade de partículas e um espaço de busca maior. Esses problemas são encontrados em várias áreas como a engenharia, biomedicina, telecomunicações e em muitas outras situações, principalmente do mundo real.

Com o advento da computação novos algoritmos afim de resolver problemas mais complexos foram desenvolvidos. Após a criação de tantos algoritmos e metaheurísticas, se faz necessário estudos para realizar a comparação de performance entre eles quando se trata de resolução de problemas em larga escala. Com esse objetivo em mente, este trabalho busca realizar a comparação entre seis algoritmos baseados em enxame de partículas.

1.1 Objetivos

Nesta seção o objetivo geral e os objetivos específicos deste trabalho são apresentados.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho consiste na realização de testes de otimização em larga escala com algoritmos de enxame de partículas que originalmente foram desenvolvidos para a resolução de problemas em pequena escala. Com esses resultados será possível definir quais algoritmos também podem ser usados para otimizar problemas em larga

escala.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Realizar o estudo bibliográfico dos algoritmos:
  - PSO - *Particle Swarm Optimization* (KENNEDY; EBERHART, 1995)
  - CCPSO2 - *New Cooperative Coevolving Particle Swarm Optimization* (LI; YAO, 2012)
  - LSSPSO - *Particle Swarm Optimization using Local Stochastic Search and Enhancing Diversity* (DING J. LIU; LEI, 2014)
  - OBL-CPSO - *Opposition-Based Learning Competitive Particle Swarm Optimizer* (ZHOU W. FANG; CHENG, 2016)
  - SPSO - *Standard Particle Swarm Optimization* (BRATTON; KENNEDY, 2007)
  - VCPSO - *Vector Coevolving Particle Swarm Optimization Algorithm* (ZHANG et al., 2017)
- Efetuar o estudo bibliográfico das funções de benchmark (JAMIL; YANG, 2013) (MOLGA; SMUTNICKI, 2005):
  - Schewfel 1.2
  - Rosenbrock
  - Sphere
  - Rastringin
  - Griewank
  - Ackley 1
- Desenvolver a pesquisa bibliográfica sobre o framework jMetal. (DURILLO; NEBRO, 2011) (NEBRO et al., 2015)
- Executar o desenvolvimento dos temas de metaherísticas e problemas em larga escala.
- Analisar os resultados de vários estudos de caso dos algoritmos já citados quando realizam a otimização de problemas em larga escala.

1.2 Motivação

A realização desse trabalho foi motivado pela dificuldade de se efetuar o planejamento de funcionamento de sistemas hidrotérmicos no território brasileiro. Por conta da grande quantidade de dados que essa atividade impõe se torna muito difícil que o planejamento seja feito de forma totalmente otimizada. Portanto, nesta seção são demonstradas as informações sobre o planejamento de sistemas hidrotérmicos no Brasil e informações sobre um simulador que visa ajudar na resolução desses problemas utilizando algoritmos de enxame de partículas.

Os resultados deste projeto visa estudar a viabilidade de utilização de algoritmos para a realização de planejamento de sistemas hidrotérmicos nesse simulador.

1.2.1 Sistemas Hidrotérmicos

Os problemas com a geração de energia no Brasil são constantes, um exemplo disso foi a falta de chuvas que atingiu o país no último ano provocando um aumento significativo no preço da energia elétrica. Portanto, o estudo de formas de otimizar a produção de energia elétrica dos sistemas hidrotérmicos é muito importante. Para se ter um sistema hidrotérmico ótimo se parte das premissas de que a quantidade de água nos reservatórios devem ser maximizadas e os custos com a geração de energia elétrica minimizados. Os sistemas hidrotérmicos são normalmente formados por hidroelétricas e termelétricas. As hidroelétricas são consideradas geradores de energia limpa e barata, porém dependem da quantidade de água armazenada nos reservatórios. As termelétricas são consideradas geradores de energia poluidoras e caras, porém não depende da disponibilidade de água, uma vez que produzem energia por meio da queima de carvão ou combustíveis fosseis.

Estudos estão sendo realizados com objetivo de otimizar o uso desses sistemas hidrotérmicos, aumentando a eficiência das hidroelétricas e termelétricas e diminuindo o custo da produção de energia. Por muitas décadas várias técnicas e métodos utilizando programação dinâmica, decomposição, redes de fluxo, inteligência artificial, dentre outras, já foram desenvolvidas para otimizar esse tipo de problema. Os algoritmos genéticos (GA) e a programação evolucionária (EP) se provaram muito eficientes resolvendo problemas complexos de sistemas de geração de energia, porém esses algoritmos não garantiam uma solução ótima perfeita (KENNEDY; SHI, 2001). Com surgimento do algoritmo de Otimização por Enxame de Partículas (PSO), que

foi inspirado no bando de pássaros, cardume de peixes e também o comportamento humano, os estudos com heurísticas modernas se tornaram mais comuns. Esse algoritmo foi desenvolvido por Kennedy e Eberhart em 1995 promovendo uma evolução enorme na solução de problemas mais complexos como o do planejamento de sistemas hidrotérmicos. Muitos algoritmos derivados do PSO foram desenvolvidos e neste estudo serão mostrados como alguns desses algoritmos funcionam e que tipo de resultados e melhoramentos eles oferecem.

1.2.2 Modelo de Otimização e Simulação dinâmicos para Sistemas Hidrotérmicos

Visando resolver o problema energético brasileiro, Ramos em 2016 publicou o seguinte trabalho que dá título a essa subseção. Este trabalho consiste em um simulador web de usinas hidrelétricas desenvolvido na linguagem de programação Java (CIGOGNA; SOARES, 1999) que simula o Sistema Integrado Nacional (SIN) (CICOGNA, 2003) e um otimizador individualizado de usinas hidrelétricas que utiliza o framework jMetal.

O processo de otimização de geração de energia elétrica de sistemas hidrotérmicos é bem complexo, portanto é considerado um problema em larga escala. O simulador foi desenvolvido para representar as usinas hidrelétricas e o otimizador utiliza de algoritmos de otimização. Em seu trabalho, Ramos utilizou o algoritmo PSO e mais três algoritmos derivados do mesmo para realizar essa otimização. Os resultados foram comparados com um outro otimizador de Programação Não Linear (PNL) (MARQUES, 2006) e não obteve resultados superiores. Porém existem muitos outros algoritmos derivados do PSO que podem ser testados e podem melhorar esse resultado significativamente.

1.3 Estrutura

A organização deste trabalho foi feito da seguinte forma: a revisão bibliográfica se encontra no próximo capítulo abrangendo metaherísticas, todos os algoritmos que foram utilizados e informações sobre o framework jMetal. No capítulo 3 todos os benchmarks que foram utilizados para testar a eficiência dos algoritmos são demonstrados. No capítulo 4 se tem os experimentos, estudos de caso e resultados. Por fim a conclusão e as referências bibliográficas.

CAPÍTULO 2

METAHEURÍSTICAS

2.1 Introdução

Em sua maioria, os problemas reais possuem dados mais complexos e são multiobjetivos, ou seja, podem ser avaliados segundo múltiplos critérios. A exemplo disso citamos o problema de otimização de sistemas hidrotérmicos que possui muitas variáveis para se levar em consideração, bem como muitos dados para serem processados, principalmente no que diz respeito a dados do passado. Os problemas em larga escala se caracterizam por conterem muitos elementos interconectados e possuírem um grande espaço de busca. Esses problemas podem ser encontrados em aplicações práticas como controle industrial, biomedicina, sistemas aeroespaciais, telecomunicações e logística. (M. ELIZABETH F. W., 2015)

Por muito tempo vários problemas em larga escala eram considerados não resolvíveis até o começo de estudos de algoritmos que solucionam problemas em larga escala. Muitos algoritmos de otimização e metaheurísticas foram desenvolvidas afim de resolver problemas em larga escala, portanto é comum que estudos e comparações sejam realizadas afim de definir qual algoritmo se faz mais adequado para resolver cada tipo de problema. Utilizando esses estudos, novos algoritmos mais robustos podem ser desenvolvidos com o objetivo final de ajudar a resolver um problema real.

O custo computacional para solucionar problemas em larga escala utilizando metaheurísticas é bem maior quando comparados a problemas simples. Muitas vezes computadores mais robustos devem ser utilizados para realizar esse tipo de processamento. Dependendo do problema a ser resolvido, a máquina irá demorar horas ou até mesmo dias para apresentar o resultado final otimizado. Em certos casos se torna possível a realização desse processamento de forma paralela utilizando sistemas distribuídos afim de diminuir o tempo de processamento.

As metaheurísticas fazem parte de um subgrupo da chamada otimização estocástica, portanto antes de falar sobre a definição de metaheurística é importante saber sobre a otimização estocástica.

A otimização estocástica consiste na busca pelo melhor elemento dentro de um conjunto de elementos disponíveis utilizando uma função objetivo definida que

apresenta o valor desejado ou um valor próximo dele. Desse modo essa técnica de otimização estuda circunstâncias em que as restrições e parâmetros dependem de váriáveis aleatórias.

A definição de metaheurística se encontra dentro do grupo de otimização estocástica, uma vez que ela se refere ao conjunto de algoritmos, técnicas e métodos que empregam algum grau de aleatoriedade para alcançar soluções ótimas de problemas considerados difíceis. Portanto, dentro do processo das metaherísticas existe uma preocupação em escapar de mínimos e máximos locais afim de encontrar uma solução ótima ou uma solução mais próxima da ótima.

Muitas vezes em problemas que se emprega o uso de metaherísticas não se sabe o valor da solução ótima, portanto se utiliza de vários testes para descobrir a melhor solução possível. Um exemplo desse tipo de problema é a programação do comportamento de um carro autônomo. Após vários testes esse carro foi programado para identificar pessoas, obstáculos e chegar em segurança da maneira mais otimizada possível de um ponto a outro, porém nunca se terá a certeza absoluta de que esse método de comportamento escolhido é a solução ótima. Segundo os vários testes realizados, essa foi considerada o melhor comportamento para o carro, entretanto nada impede que novos estudos sejam feitos e um novo comportamento tenha melhores resultados que esse.

Por fim, se é entendido que metaheurísticas são utilizadas para resolver problemas em que se tem pouca informação, seu espaço de busca é grande e a força bruta não é considerada, ou seja, não se sabe como é a solução ótima, porém é possível que após vários testes a melhor solução seja considerada. Um vantagem é a possibilidade de testes sobre as soluções encontradas, ou seja, é possível realizar alterações no modo que a metaheurística foi aplicada e comparar esse novo resultado com o anterior.

2.2 Algoritmos de Otimização por Enxame de Partículas

Neste capítulo serão apresentados vários algoritmos heurísticos de otimização por enxame de partículas. Todos esses algoritmos são derivados do PSO, ou seja, mudanças foram acrescentadas nesse algoritmo "pai" com o objetivo de melhorar a performance ou resolver outros tipos específicos de problemas.

2.2.1 PSO

O primeiro método apresentado neste trabalho é o algoritmo de Otimização por Enxame de Partículas abreviado por PSO (do inglês *Particle Swarm Optimization*). O PSO é um algoritmo de otimização projeto para resolução de problemas não-lineares, contínuos ou discretos. Ele foi inspirado no comportamento de bando de pássaros, cardume de peixes, enxames de abelhas etc. Sua principal característica é a simplicidade, ele foi desenvolvido utilizando operações matemáticas primitivas e não necessita de grande poder computacional para executar.

A hipótese fundamental para o desenvolvimento do PSO leva em consideração a seguinte vertente. Foi entendido que em bando de pássaros e cardumes de peixes os animais compartilhavam informações sociais entre eles. Uma analogia interessante citada por (WILSON, 1975) mostra que se alguém jogar vários grãos de milho em algum local aberto, temos a certeza que em algum momento vários pássaros vão estar lá se alimentando. Eles procuram comida aleatoriamente indo de árvore em árvore, até acharem comida e de alguma forma todo o seu bando fica sabendo disso.

Na prática o PSO define um espaço de busca multidimensional que será explorado pelo enxame. Inicialmente, as partículas são inicializadas em posições aleatórias dentro desse espaço definido. Essas posições são representadas pelo vetor $\overrightarrow{X}_i = (X_{i1}, X_{i2}, ..., X_{id})$, onde cada variável possui uma posição específica de acordo com o problema. Além da posição, cada partícula também tem uma velocidade inicializada de forma aleatória. As velocidades das partículas são representadas pelo vetor $\overrightarrow{V}_i = (V_{i1}, V_{i2}, ..., V_{id})$. Após cada iteração todas as partículas passam a ter uma nova posição. A atualização da posição das partículas é feita seguindo a equação 2.1.

$$\overrightarrow{X}_{id}(t+1) = \overrightarrow{X}_{id}(t) + \overrightarrow{V}_{id}(t+1), \tag{2.1}$$

Onde a variável t é o índice da iteração, $i = 1, ..., N_p$, sendo N_p o número de partículas do bando e $d = 1, ..., N_g$, sendo N_g o número de parâmetros ou dimensões de uma partícula.

O conceito da busca pelo local ótimo se dá pela alteração do vetor velocidade de cada partícula em cada iteração, assim atualizando para uma nova posição.

Basicamente cada partícula irá ter sua posição ajustada de acordo com a influência de outra partícula mais bem posicionada. Cada partícula possui um valor que mostra o quão próximo do resultado ideal ela está, esse valor pode ser calculado de várias formas dependendo do problema que está sendo resolvido. Portanto, cada problema possui uma funcão fitness que define esse valor.

Existem duas maneiras comuns de definir qual partícula vai influenciar as outras em cada iteração. Essa definição é feita pela topologia escolhida. As duas topologias mais comuns são a anel e a estrela e estão representadas na figura 2.1. Na topologia anel cada elemento só tem a informação de seus vizinhos, portanto só podem ser influenciados por uma das duas partículas vizinhas a cada iteração. Representamos a posição da melhor partícula vizinha por $\overrightarrow{N}_{best_i}$ ou *local best*. Na topologia estrela todas as partículas se comunicam entre si, portanto a posição da melhor partícula é representada por $\overrightarrow{G}_{best}$ ou *global best* e ela influencia toda a população.

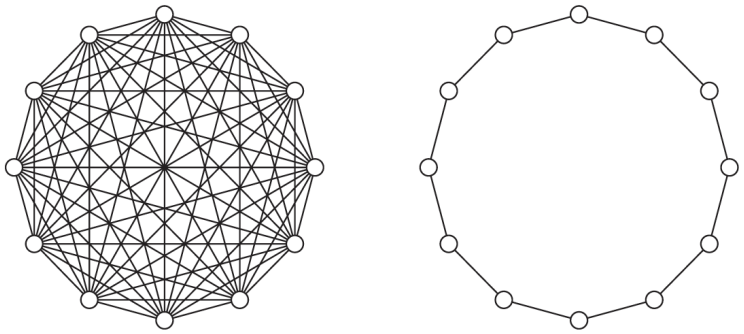

Figura 2.1 - Topologia: (a) Estrela e (b) Anel

O ajuste da velocidade de cada partícula é feito conforme a equação 2.2.

$$\begin{aligned}\overrightarrow{V}_{id}(t+1) = \omega \cdot \overrightarrow{V}_{id}(t) \\ + c_1 \cdot rand[0,1] \cdot [\overrightarrow{P}_{best_i}(t) - \overrightarrow{X}_{id}(t)] \\ + c_2 \cdot rand[0,1] \cdot [\overrightarrow{N}_{best_i}(t) - \overrightarrow{X}_{id}(t)],\end{aligned} \tag{2.2}$$

onde:

- ω - fator de inércia;
- c_1 - uma constante representando o coeficiente cognitivo;
- c_2 - uma constante representando o coeficiente social;
- $rand[0,1]$ - valor aleatório distribuído de forma uniforme no intervalo [0,1];
- $V_{id}(t)$ - velocidade da partícula i na iteração t;
- $\overrightarrow{P}_{best_i}(t)$ - melhor posição da partícula i encontrda até a iteração t;
- $\overrightarrow{N}_{best_i}(t)$ - posição da partícula vizinha mais bem posicionada;

Uma observação importante a se fazer sobre a equação 2.2 é que quando a topologia utilizada for a estrela, o $\overrightarrow{N}_{best}$ é substituído por $\overrightarrow{G}_{best}$.

A seleção dos parâmetros no PSO é de extrema importância, pois eles são um dos principais responsáveis por facilitar a convergência e evitar que o enxame 'exploda". As constantes c_1 e c_2 existem para manter uma paridade entre a posição da partícula em questão e a posição da partícula mais bem posicionada. O valor mais utilizado para esses parâmetros é 1,49445.

A velocidade $\overrightarrow{V}_{id}(t)$ normalmente possui limites definidos para evitar o distanciamento brusco para locais indesejados quando a velocidade é muito grande e também a falta de exploração quando essa velocidade é muito baixa. Comumente os limite são definidos seguindo a lógica $V_d^{min} \leq V_{id}(t) \leq V_d^{max}$ variando de problema para problema.

O fator de inércia ω também segue uma lógica de atualização representada pela equação 2.3, no qual ω_{max} é o fator de inércia máximo, ω_{min} é o fator de inércia mínimo, $iter_{max}$ é a quantidade máxima de iterações do experimento e por fim t é a iteração atual.

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{iter_{max}} \cdot t \tag{2.3}$$

O pseudocódigo do algortimo PSO descrito nessa seção é representado pelo algoritmo 1.

Algoritmo 1: Pseudocódigo do algoritmo PSO

```
1  início
2      Atribuição de valores para os parâmetros;
3      i = 0;
4      repita
5          Inicializa X⃗_i com uma posição aleatória;
6          Inicializa V⃗_i com uma velocidade aleatória;
7          i++;
8      até i = tamanhoDoEnxame;
9      t = 0;
10     repita
11         i = 0;
12         repita
13             Atualizar V⃗_i utilizando a equação 2.2;
14             Atualizar X⃗_i utilizando a equação 2.1;
15             Atualizar P⃗_best e L⃗_best;
16             i++;
17         até i = tamanhoDoEnxame;
18         t++;
19     até t atingir iter_max;
       Saída: melhor L⃗_best
20 fim
```

2.2.2 CCPSO2

O primeiro algoritmo derivado do PSO a ser apresentado nesta dissertação é o Novo Algoritmo de Enxame de Partículas Cooperativo com Coevolução (CCPSO2) do inglês *New Cooperative Coevolving Particle Swarm Optimization*. O intuito do desenvolvimento desse algoritmo é resolver problemas em larga escala com muitas variáveis, uma vez que os problemas do mundo real estão cada vez mais complexos. A maioria das metaheurísticas derivadas de algoritmos evolucionários e de enxame de partículas tem uma baixa performance em problemas de larga escala.

O CCPSO2 é derivado do algoritmo CCPSO desenvolvido por (LI; YAO, 2009) e utiliza de várias artimanhas para melhorar a performance e confiabilidade em relação ao seu antecessor. Primeiramente, se utiliza as distribuições de Cauchy e Gauss para gerar a próxima posição das partículas, ao contrário do adotado pelo PSO que utiliza o termo de velocidade. Para garantir uma convergência mais lenta e uma maior diversidade da população foi usada a topologia anel.

Baseado nessas definições a regra de atualização de posição é dada pela fórmula a seguir:

$$X_{i,d}(t+1) = \begin{cases} y_{i,d}(t) + C(1)|y_{i,d}(t) - \hat{y}'_{i,d}(t)|, & se\ rand \leq p \\ \hat{y}'_{i,d}(t) + N(0,1)|y_{i,d}(t) - \hat{y}'_{i,d}(t)|, & senao \end{cases} \tag{2.4}$$

onde:

- $C(1)$ - número gerado pela distribuição Cauchy;
- $|y_{i,d}(t) - \hat{y}'_{i,d}(t)|$ - desvio padrão efetivo para as distribuições Gaussiana e Cauchy;
- $rand$ - um número entre 0 e 1 gerado randomicamente;
- p - um valor derivado da distribuição de Cauchy (normalmente se utiliza 0.5);
- $\hat{y}'_{i,d}$ - *lbest*

Utilizando a topologia anel o *lbest* é definido comparando três partículas, a partícula atual e seus dois vizinhos. Essa topologia previne a convergência prematura.

Outro mecanismo utilizado no algoritmo CCPSO2 é a mudança do tamanho do grupo de partículas de forma dinâmica. Ao invés de utilizar um grupo com tamanho fixo, um mecanismo de grupo aleatório é acionado, portanto em cada iteração o tamanho do grupo é escolhido de forma randômica. A ideia é que uma variável s que representa o tamanho do grupo seja escolhida de forma randômica dentro de um conjunto de valores S. Em cada iteração o valor $fitness$ é gravado, se não houver uma melhoria um novo s é escolhido, caso tenha melhoria o valor de s se mantém o mesmo. Esse tipo de abordagem faz com que o procedimento não fique preso em um mesmo valor, o que prejudicaria a busca pela otimização.

Dois loops são utilizados na iteração, um para cada enxame e um para cada partícula. Primeiramente existe a checagem da melhor partícula $P_j.x_i$ dentro de um enxame específico e depois a checagem do melhor enxame $P_j.\hat{y}$. Utilizando o conceito de *localbest* $P_j.\hat{y}'_i$ que é a melhor partícula do melhor subgrupo é definida. No segundo loop ocorre o cálculo da equação 2.4.

Com o objetivo de atualizar os valores individuais de cada partícula, duas matrizes são utilizadas para armazenar a atual e a melhor posição de todos os enxames. Utilizando permutações aleatórias o valor $fitness$ por coevolução de cada nova partícula é obtido.

O resumo do funcionamento do CCPSO2 é representado pelo pseudocódigo 2

Algoritmo 2: Pseudocódigo do algoritmo CCPSO2

início
  Criar e inicializar K enxames com s dimensões (s é escolhido de forma aleatória dentro de um conjunto de valores definidos S e $n = K * s$ é sempre verdadeiro);
  O enxame jth é dado como $P_j, j \in [1..K]$;
  repita
    se $f(\hat{y})$ *não melhorar* **então**
      Escolha s de forma randômica;
      $K = n/s$;
      Permute todos o índices das n dimensões;
      Construa K enxames em cada dimensão s;
    para *Cada enxame* $j \in [1..K]$ **faça**
      para *Cada partícula* $i \in [1..TamEnx]$ **faça**
        se $f(b(j, P_j.x_i)) < f(b(j, P_j.y_i))$ **então**
          $P_j.y_i \leftarrow P_j.x_i$;
        se $f(b(j, P_j.y_i)) < f(b(j, P_j.\hat{y}))$ **então**
          $P_j.\hat{y} \leftarrow P_j.y_i$;
      fim
      para *Cada partícula* $i \in [1..TamEnx]$ **faça**
        $P_j.y'_i \leftarrow localBest(P_j.y_{i-1}, P_j.y_i, P_j.y_{i+1})$;
      fim
      se $f(b(j, P_j.\hat{y})) < f(\hat{y})$ **então**
        jth parte de $\hat{y}$ é substituída por $P_j.\hat{y}$;
    fim
    para *Cada enxame* $j \in [1..K]$ **faça**
      para *Cada partícula* $i \in [1..TamEnx]$ **faça**
        Atualiza a posição da partícula ith no enxame P_j utilizando a equação 2.4;
      fim
    fim
  até *Atingir critério de parada*;
fim

2.2.3 LSSPSO

A próxima variação do PSO a ser abordada neste trabalho é o LSSPSO (*Particle Swarm Optimization using Local Stochastic Search and Enhancing Diversity*) desenvolvido por (DING J. LIU; LEI, 2014) que foi inspirada em um fenômeno social, no qual o indivíduo busca se destacar primeiramente entre seus vizinhos e depois se destacar perante toda a comunidade.

Utilizando essa premissa, cada partícula possui parâmetros individuais de inércia e aceleração e eles variam durante as iterações. O objetivo dessa variação é aumentar a diversidade e prevenir que ocorra uma convergência prematura.

Esse algoritmo utiliza um mecanismo capaz de aumentar a diversidade da população. Esse mecanismo se evita a necessidade de cálculo de diversidade e mantém a diversidade do enxame. Para cada partícula $x_i(t)$, uma partícula $x_i(t+1)$ é criada pelas equações de velocidade e posição já vistas na seção do PSO. Recombinando $x_i(t)$ e $x_i(t+1)$ uma terceira partícula $px_i(t+1)$ foi gerada da seguinte forma:

$$px_i(t+1) = \begin{cases} x_i(t+1), & se\, f(x_i(t+1)) < f(x_i(t)) \wedge rand(0,1) < pr \\ x_i(t), & senão \end{cases} \tag{2.5}$$

$$pv_i(t+1) = v_i(t+1) \tag{2.6}$$

onde:

- i = 1,2,...,N;
- $f(.)$ - avaliação da função *fitness*;
- $rand[0,1]$ - valor aleatório distribuído de forma uniforme no intervalo [0,1];
- pr - uma constante predefinida (normalmente se utiliza o valor 0.5);

A constante pr é responsável pela similaridade entre $x_i(t+1)$ e $px_i(t+1)$, no qual quanto mais alto o valor de pr maior é a similariedade entre elas.

Após a recombinação o mecanismo de seleção guloso é utilizado:

$$x_i(t+1) = \begin{cases} px_i(t+1), & se\, px_i(t+1) < x_i(t) \\ x_i(t), & senão \end{cases} \tag{2.7}$$

No algoritmo PSO as partículas tendem a mover para a mesma direção, assim após cada iteração as partículas se tornam cada vez mais parecidas. Ao utilizar uma terceira partícula $px_i(t+1)$ essas semelhanças diminuem, assim ocorre o aumento da diversidade da população.

Uma convergência prematura acontece quando a maioria das partículas param de melhorar suas posições mesmo após uma sucessão grande de interações. Esse problema normalmente acontece quando se escolhe um valor muito pequeno para a inércia ω ou quando os valores dos parâmetros não são escolhidos da melhor forma. Com o objetivo de prevenir que isso aconteça, o LSSPSO utiliza de uma estratégia chamada de busca local estocástica. Essa estratégia foi baseada nos Jogos Olímpicos, no qual o melhor atleta de um país compete com outros bons atletas, ou seja, o atleta primeiramente foi o melhor em seu país para depois tentar se tornar o melhor do mundo. A figura 2.2.3 mostra a busca do ótimo local como sugerido na abordagem apresentada.

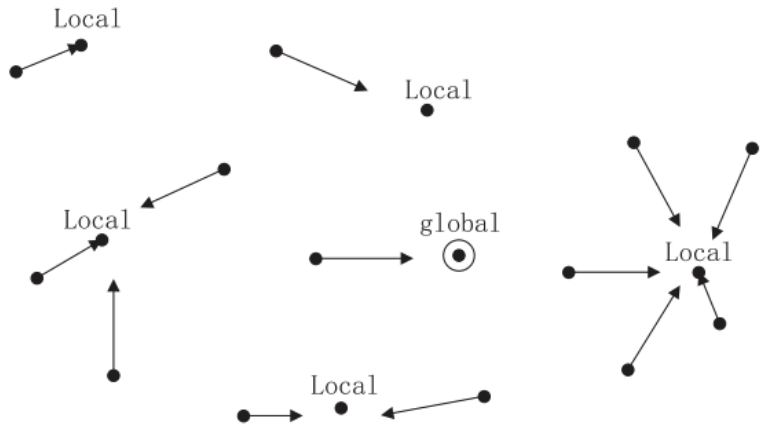

Figura 2.2 - Busca do ótimo local.

Durante a busca do ótimo local, a partícula continuará seguindo a mesma

direção e não irá alterá-la por influência de outra partícula como normalmente acontece. A velocidade das partículas são atualizadas de acordo com a fórmula 2.8 e por fim a variável temporária $LSSx_i(t+1)$ é atualizada seguindo a equação 2.9.

$$v_i(t+1) = \begin{cases} \omega\, rand(0,1)v_i(t), & se\, x_i(t) = gbest \\ \omega\, rand(0,1)v_i(t) + r\, randV, & senão \end{cases} \tag{2.8}$$

$$LSSx_i(t+1) = x_i(t) + v_i(t+1) \tag{2.9}$$

onde r é uma constante com valor pequeno e $randV$ é um vetor unitário aleatório.

No próximo passo a variável $LSSx_i(t+1)$ é comparada com $x_i(t+1)$ que foi calculada utilizando a fórmula clássica do PSO. Quem tiver a melhor *fitness* suprecede a outra, ou seja, se $LSSx_i(t+1)$ se sair melhor que $x_i(t+1)$ a $i$ésima partícula continuará se movendo na mesma direção durante as iterações até seu valor parar de melhorar.

A metaheurística LSSPSO utiliza dessas duas técnicas para compor sua estratégia de otimização. Esse algoritmo tem seu funcionamento mostrado de forma detalhada no pseudocódigo 3.

Algoritmo 3: Pseudocódigo do algoritmo LSSPSO

```
1  início
2      Inicializar cada partícula do enxame de forma randômica;
3      Inicializar pbest e gbest;
4      repita
5          para i = 1 até N faça
6              Calcular velocidade da partícula x_i de acordo com a equação 2.2;
7              Atualizar posição da partícula x_i de acordo com a equação 2.1;
8              Calcular o valor fitness de x_i;
9              Gerar a nova partícula px_i de acordo com as equações 2.5 e 2.6;
10             Calcular o valor fitness de px_i;
11             Seleção gulosa de acordo com a equação 2.7;
12             Atualizar pbest_i e gbest;
13         fim
14         para i = 1 até N faça
15             repita
16                 Gerar LSSx_i de acordo com as equações 2.8 e 2.9;
17                 se f(LSSx_i) ≤ f(x_i) então
18                     x_i = LSSx_i;
19                 senão
20                     Sair do Repita;
21                 fim
22             até;
23             Atualizar pbest_i e gbest;
24         fim
25         Iter++;
26     até Max ≤ Iter;
27 fim
```

2.2.4 OBL-CPSO

O algoritmo OBL-CPSO (*Opposition-Based Learning Competitive Particle Swarm Optimizer*) foi proposto com o objetivo de resolver a convergência prematura do PSO. Dois mecanismos foram utilizados na criação desse algoritmo, a aprendizagem baseada em oposição e o otimizador de enxame competitivo.

A aprendizagem baseada em oposição foi proposta por (TIZHOOSH, 2005) e se mostrou muito eficiente na evolução diferencial (RAHNAMAYAN et al., 2008). Um ponto importante na aprendizagem baseada em oposição é que o candidato a solução e seu oposto correspondente podem ser calculados em uma única iteração. Podemos afirmar que a partícula oposta é uma "extensão" da partícula original. O cálculo da partícula oposta $\widetilde{x}$ se dá utilizando a equação 2.10 sendo x um número real pertencente ao intervalo $[a, b]$.

$$\widetilde{x} = a + b - x \tag{2.10}$$

Estendendo esse conceito para uma espaço multidimensional é simples. Sendo $x_1, x_2, ..., x_D$ números reais e x_i pertencente ao intervalo $[a, b]$, a partícula $\widetilde{P}$, que é a partícula oposta de $P(x_1, x_2, ..., x_D)$, é calculada utilizando as coordenadas opostas $\widetilde{x}_1, \widetilde{x}_2, ..., \widetilde{x}_D$ seguindo a equação 2.11.

$$\widetilde{x}_i = a_i + b_i - x_i \tag{2.11}$$

Basicamente a vantagem de se utilizar a aprendizagem baseada em oposição é que a busca é feita em um ponto e no seu oposto ao mesmo tempo, caso o fitness seja melhor no lado oposto, a partícula pula diretamente para o oposto ou para sua vizinhança. Portanto, o espaço de busca é percorrido mais rapidamente aumentando a probabilidade de encontrar o ponto ótimo.

Um dos maiores motivos para convergência prematura no PSO é a influência que o *gbest* tem sobre o enxame. Através do processo evolucionário o melhor local que cada partícula obteve tem um impacto parecido com o impacto do *gbest*. Com o objetivo de diminuir a influência do *gbest* ou até mesmo excluir essa influência, o algoritmo CSO (Competitive Swarm Optimization) (CHENG; JIN, 2014) foi desenvolvido. Esse

algoritmo coloca a ideia de competição entre as partículas, no qual partículas são combinadas de forma aleatória e participam de uma competição que terá partículas vencedoras e perdedoras. As partíclas vencedoras passam para a próxima interação sem alteração e as perdedoras são influenciadas pelas respectivas vencedoras de sua competição.

No algoritmo OBL-CPSO cada competição é formada por três partículas escolhidas aleatoriamente no enxame. Analizando o valor $fitness$ de cada partícula se tem a partícula vencedora, perdedora e a mediana. A partícula vencedora passa sem alterações para a próxima iteração, a partícula perdedora sofre influência da vencedora através do mecanismo competitivo de aprendizado antes de passar para a próxima iteração. A partícula que não foi a vencedora nem a perdedora vai utilizar o mecanismo de aprendizagem baseada em oposição e passar para a próxima iteração, assim ajudando a explorar todo o espaço de busca. Esse princípio é detalhado na figura 2.2.4.

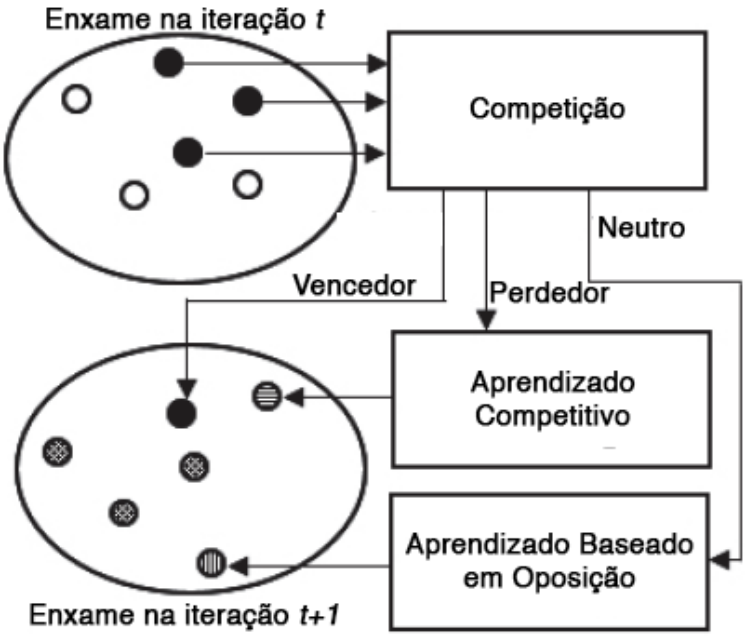

Figura 2.3 - Princípios do OBL-CPSO. (ZHOU W. FANG; CHENG, 2016)

As velocidades e posições da partícula perdedora e neutra são atualizadas segundo as equações 2.12, 2.13 e 2.14:

$$V_{ld}^{k}(t+1) = R_{1d}^{k} \cdot V_{ld}^{k}(t) + R_{2d}^{k} \cdot (X_{wd}^{k}(t) - X_{ld}^{k}(t)) + \varphi \cdot R_{3d}^{k}(t) \cdot (\overline{X}_{d}^{k}(t) - X_{ld}^{k}(t)) \quad (2.12)$$

$$X_{ld}^{k}(t+1) = X_{ld}^{k} + V_{ld}^{k}(t+1) \quad (2.13)$$

$$X_{nd}^{k}(t+1) = ub_d + lb_d - X_{nd}^{k}(t) + R_{4d}^{k}(t) \cdot X_{nd}^{k}(t) \quad (2.14)$$

onde:

- k - $round$ da competição;d;
- V_{ld}^{k} - dth velocidade da partícula perdedora;
- X_{wd}^{k} - dth posição da partícula vencedora;
- X_{nd}^{k} - dth posição da partícula mediana;
- X_{ld}^{k} - dth posição da partícula perdedora;
- t - iteração;
- R_{1d}^{k}, R_{2d}^{k}, R_{3d}^{k}, R_{4d}^{k} - números randômicos pertencentes ao intervalo [0,1];
- φ - parâmetro manual;
- $\overline{X}_{d}^{k}(t)$ - posição média das partículas;
- ub_d - limite superior do espaço de busca;
- lb_d - limite inferior do espaço de busca;

O pseudocódigo da metaheurística OBL-CPSO é representado pelo algoritmo 4.

Algoritmo 4: Pseudocódigo do algoritmo OBL-CPSO

```
1  início
2      Inicializar P;
3      repita
4          para k = 1 até N/3 faça
5              //Escolha três partículas do enxame;
6              r_1 = P(k);
7              r_2 = P(k + N/3);
8              r_3 = P(k + 2N/3);
9              //Realize a competição das três partículas;
10             (w, n, l) = competir(r_1, r_2, r_3);
11             Atualizar X_ld^k de acordo com as equações 2.12 e 2.13;
12             Atualizar X_nd^k de acordo com a equação 2.14;
13             Atualizar o fitness da partícula perdedora e neutra;
14         fim
15     até Atingir critério de parada;
16 fim
```

2.2.5 SPSO

Com o intuito de definir um padrão específico para testes com o PSO, o *Standard Particle Swarm Optimization* ou SPSO foi desenvolvido. Esse algoritmo é o PSO original com padrões especificados. O motivo dessa padronização é viabilizar a comparação entre os estudos que utilizam o PSO, uma vez que existem diversas formas e padrões para rodar o PSO.

Os padrões utilizados no SPSO são:

A topologia utilizada é a topologia anel, representada na figura 2.4, em que se usa o *localbest*, uma vez que possui uma convergência mais lenta em relação a topologia estrela. Uma convergência rápida significa alta performance, porém o resultado em problemas mais complexos são piores do que quando utilizado uma convergência mais lenta.

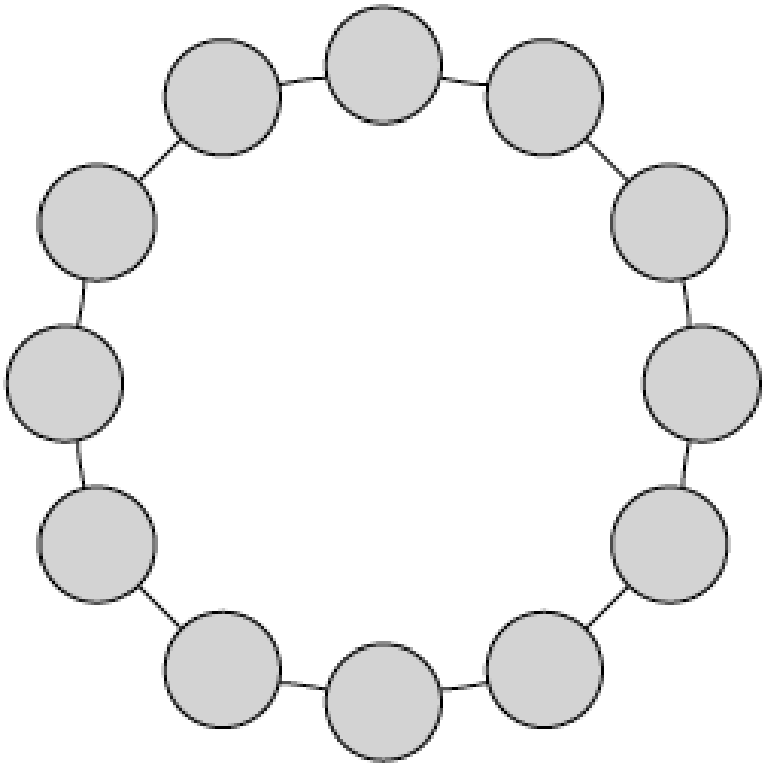

Figura 2.4 - Topologia Anel

A velocidade de cada partícula é representada pela equação 2.15.

$$\overrightarrow{v}_{id} = \chi(wv_{id} + c_1\epsilon_1(p_{id} - x_{id}) + c_2\epsilon_2(p_{gd} - x_{id})) \tag{2.15}$$

A posição de cada partícula é representada pela equação 2.16

$$x_{id} = x_{id} + v_{id} \tag{2.16}$$

A inicialização do enxame é realizada de forma não uniforme. A quantidade de partículas usada nessa padronização é 50, porém como esse trabalho visa avaliar algoritmos para situações de grande quantidade de partículas, essa regra não foi utilizada. Por fim, partículas que estão foram do espaço de busca não são avaliadas.

2.2.6 VCPSO

Finalmente, o último algoritmo estudado é o algoritmo de Otimização de Enxame de Partículas utilizando Coevolução de Vetores ou VCPSO (*Vector Coevolving Particle Swarm Optimization Algorithm*). Esse algoritmo também é derivado do PSO e foi modificado de várias formas buscando um melhor resultado.

A primeira característica dessa metaheurística é a técnica de particionamento vetorial aleatório. Esse método divide randomicamente dimensões completas em pequenos segmentos. A figura 2.5 representa a partição vetorial.

Em um problema de dimensão D, a dimensão completa de cada partícula é particionada de forma randômica em $k+1$ segmentos com k pontos de divisão. Esses índices de pontos de divisão são gerados de forma randômica obedecendo a equação 2.17.

$$S_i = Rand(S_{i-1} + m, Dim - 1 - m) \tag{2.17}$$

onde:

- S_i - índice atual do ponto de divisão;
- S_{i-1} - índice do ponto de divisão anterior;
- m - intervalo mínimo de cada segmento;

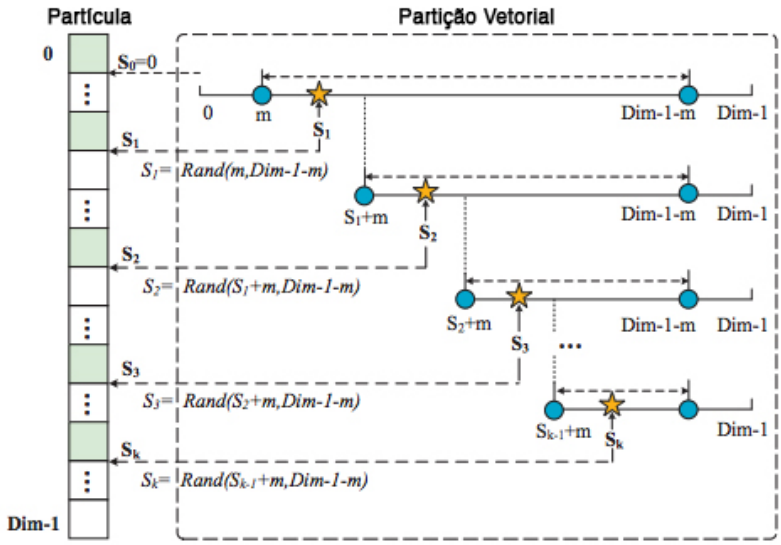

Figura 2.5 - Partição Vetorial (ZHANG et al., 2017)

Portanto, a fórmula 2.17 retorna um valor inteiro aleatório dentro do intervalo $S_{i-1} + m, Dim - 1 - m$. Cada segmento contém pelo menos m elementos e o número total de pontos de divisão satisfaz a regra $k \in [1, [D/m]]$.

Os detalhes dessa operação é mostrado no algoritmo 5.

Algoritmo 5: Pseudocódigo da Partição Vetorial

```
1  início
2      Inicializar k_i = 1;
3      para j = 0 até D faça
4          S_i[j] = 0;
5      fim
6      para j = 1 até D faça
7          Star = S_i[j − 1] + 1;
8          se Star ≥ D − 1 − m então
9              break;
10         senão
11             end = Rand(Start, D − 1 − m);
12             S_i[j] = end;
13             k_i = k_i + 1;
14         fim
15     fim
16 fim
```

Após esse particionamento os elementos adjacentes de cada sub-dimensão podem ter três tipos de relação:

- Relação Melhor;
- Relação Equivalente;
- Relação Pior;

Essas relações podem ser visualizadas de forma abstrata como movimentos físicos para uma posição mais alta, mais baixa ou a combinação das duas. Portanto, alguns operadores foram criados para esses movimentos físicos das partículas.

- Operador Crescente;
- Operador Decrescente;
- Operador Colina;
- Operador Lago;

- Operador Centralizado;
- Operador Descentralizado;

A definição dos operadores é representada pela figura 2.6.

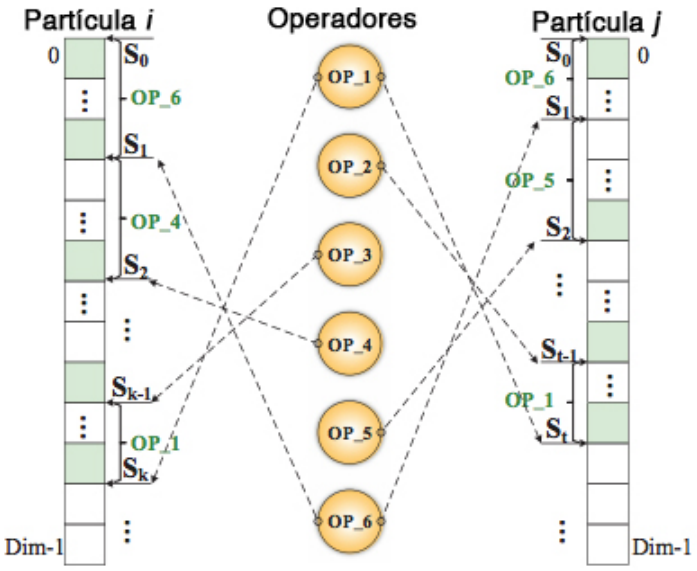

Figura 2.6 - Definição dos Operadores (ZHANG et al., 2017)

O Operador Crescente busca aumentar a relação entre dimensões adjacentes no mesmo segmento. Por exemplo, para a jth dimensão de uma partícula i, se o valor de x_i^j é menor que o da sua dimensão anterior x_i^{j-1}, o valor da jth dimensão irá aumentar seguindo a equação 2.18. Em um problema de larga escala a partícula irá mover para uma posição relativa mais alta.

$$x_i^j(t+1) = x_i^j(t) + \Phi \cdot (x_i^{j-1}(t) - x_i^j(t)), j \in [s, e] \tag{2.18}$$

Onde Φ é um número aleatório gerado da distribuição uniforme $U(0,1)$, s é o índice inicial do segmento correspondente e e o índice final.

O Operador Decrescente é o inverso do Operador Crescente. Ele busca diminuir a relação entre dimensões adjacentes no mesmo segmento. Em um problema de larga escala a partícula irá mover para uma posição relativa mais baixa. Esse operador

segue a lógica da equação 2.19.

$$x_i^j(t+1) = x_i^j(t) - \Phi \cdot (x_i^j(t) - x_i^{j-1}(t)), j \in [s, e] \tag{2.19}$$

O Operador Colina e o Operador Lago são operadores que buscam mesclar os relacionamentos com as características dos Operadores Crescente e Decrescente. Um ponto de divisão é gerado de forma aleatória. O Operador Colina aplica o Operador Crescente na primeira parte e o Operador Decrescente na segunda. O Operador Lago faz o contrário, aplica o Operador Decrescente na primeira parte e o Operador Crescente na segunda parte.

O Operador Centralizado representado na figura 2.7(a) busca realçar a habilidade de busca e acelerar a convergência do VCPSO. Ele busca o ponto médio entre as k melhores partículas individuais. Assim esse ponto é dado como muito promissor, portanto o segmento influenciado por esse operador segue as equações 2.20 e 2.21.

$$v_i^j(t+1) = wv_i^j(t) + c * r * (Center_i^j(t) - x_i^j(t)) \tag{2.20}$$

$$X_i^j(t+1) = x_i^j(t) + v_i^j(t+1) \tag{2.21}$$

onde:

- $w = 0.721$;
- $c = 2$;
- r - número aleatório distribuido uniformemente no intervalo $[0, 1]$;

O Operador Descentralizado representado na figura 2.7(b) busca realçar a habilidade de busca. Sua operação é oposta ao Operador Centralizado, no qual a sub-dimensão jth é atualizada conforme a partícula estocástica $\gamma_i(j)$ que possui um $fitness$ do $pbest$ melhor que outra partícula selecionada aleatoriamente. Esse operador segue as equações 2.22 e 2.23.

$$v_i^j(t+1) = wv_i^j(t) + c * r * (Pbest^j_{\gamma_i(j)}(t) - x_i^j(t)) \tag{2.22}$$

$$X_i^j(t+1) = x_i^j(t) + v_i^j(t+1) \tag{2.23}$$

onde:

- $w = 0.6$;
- $c = 1$;
- r - número aleatório distribuido uniformemente no intervalo $[0, 1]$;

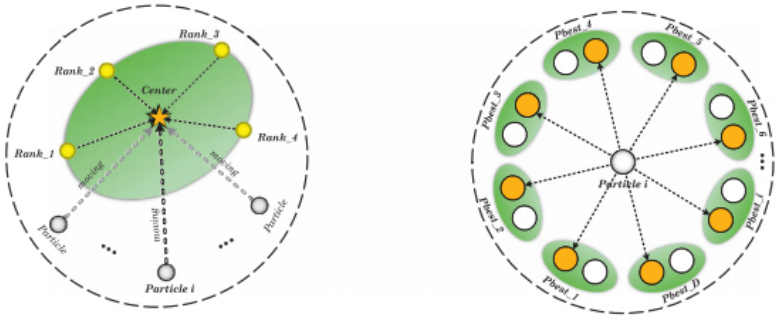

Figura 2.7 - (a) Operador Centralizado e (b) Operador Descentralizado

O funcionamento do algoritmo VCPSO está representado no pseudocódigo 6.

Algoritmo 6: Pseudocódigo do algoritmo VCPSO

```
1  início
2      Inicializar t = 1;
3      para i = 1 até M faça
4          para j = 1 até D faça
5              Inicializar velocidade e posição de cada partícula;
6          fim
7          Particionamento Vetorial 5;
8          Definir operadores para cada segmento;
9      fim
10     repita
11         para i = 1 até M faça
12             f(x_i(t)) = fitness da partícula x_i(t);
13             Atualizar posição pbest_i da partícla i;
14             Atualizar posição gbest_i da iteração tth;
15         fim
16         para i = 1 até M faça
17             se PbestNoUpdateTimes_i > 2 então
18                 Particionamento Vetorial 5;
19                 Definir operadores para cada segmento;
20                 PbestNoUpdateTimes_i = 0;
21             para k = 1 até k_i faça
22                 Atualizar X_i e V_i com os operadores correspondentes;
23             fim
24         fim
25         t = t + 1;
26     até (|f(x(t) * | ≥ ε)or(t ≤ T);
27     Retorne a melhor posição x * (t);
28 fim
```

2.3 jMetal

O desenvolvimento de algoritmos de metaheurística se tornam bem complexos quando são escritos em uma linguagem de programação que não seja robusta. A linguagem de programação escolhida foi o Java. O Java é uma linguagem de programação orientada a objetos que é famosa por funcionar em diferentes ambientes. Uma importante premissa é que o java é compatível com os sistemas operacionais mais utilizados do mercado.

Com o objetivo de facilitar a integração dos algoritmos com outros sistemas o framework jMetal (NEBRO et al., 2015) que é específico para resolver esses tipos de problemas foi escolhido.

O framework jMetal (DURILLO; NEBRO, 2011) pode ser utilizado para trabalhar com experimentos que fornecem técnicas do estado da arte, para criar os próprios algoritmos (o caso deste trabalho), resolver problemas de otimização, integração de ferramentas etc. Segundo informações do site do jMetal, a motivação para a criação desse framework se deu para que os próprios desenvolvedores tivessem uma facilidade maior em trabalhar com pesquisa de otimizações multi-objetivas. Portanto, os desenvolvedores procuraram criar um software que fosse fácil de usar. O jMetal está em constante evolução, no qual janeiro de 2014 foi lançada a versão 4.5 do jMetal e a última atualização até a produção deste trabalho foi em dezembro de 2017. Este trabalho foi desenvolvido utilizando a versão 5.1 do jMetal.

CAPÍTULO 3

FUNÇÕES DE BENCHMARK

3.0.1 Introdução

Os testes de confiabilidade, eficiência e validação da otimização utilizando algoritmos são realizados executando repetidamente usando um conjunto de padrões de referência ou funções de teste na literatura(JAMIL; YANG, 2013). Portanto, a performance de cada algoritmo é medida quando executados utilizando benchmarks que são comumente utilizados em trabalhos científicos de otimização de algoritmos (MOLGA; SMUTNICKI, 2005).

Seis funções de benchmark foram escolhidas para a realização dos testes de qualidade dos algoritmos desenvolvidos. Os mínimos ótimos de todas as funções é zero.

Essas funções oferecem a possibilidade de comparar o desempenho dos algoritmos correlacionando características de cada função. Por exemplo, a modalidade de uma função é correlacionada ao número de máximos ou mínimos presentes no seu gráfico plotado e pode ter um impacto adverso nas buscas pelo ponto ótimo. Portanto, essas funções apresentam desafios para a convergência dos algoritmos.

3.0.2 Schwefel 1.2

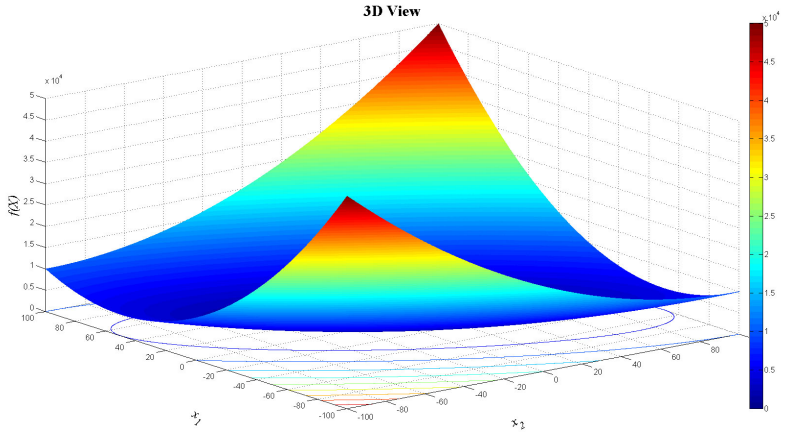

Figura 3.1 - Função Schwefel 1.2

A função Schwefel 1.2 é representada pela figura 3.1 e é caracterizada por $f(x_1 \cdots x_n) = \sum_{i=1}^{D}(\sum_{j=1}^{i} x_j)^2$ sujeito a $-100 \leq x_i \leq 100$ com mínimo global alocado em $x* = f(0, \cdots, 0)$, $f(x*) = 0$.

3.0.3 Rosenbrock

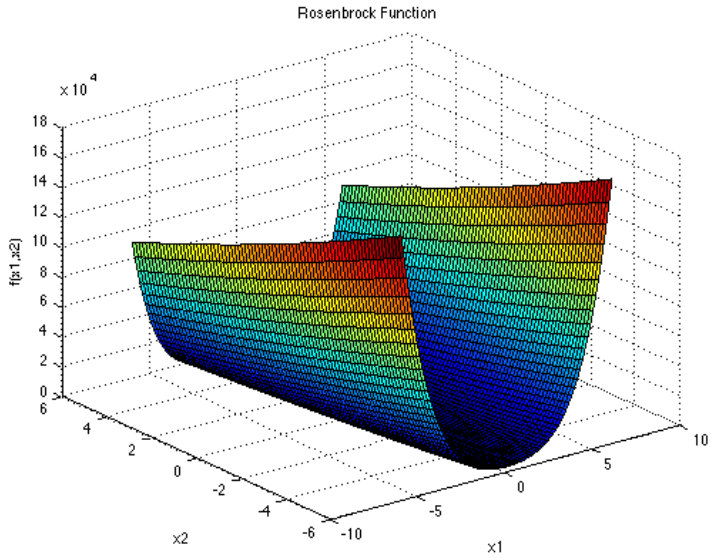

Figura 3.2 - Função Rosenbrock

A função Rosenbrock é representada pela figura 3.2 e é caracterizada por $f(x_1 \cdots x_n) = \sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$ sujeito a $-2.048 \leq x_i \leq 2.048$ com mínimo global alocado em $f(1, 1, \cdots, 1) = 0$.

3.0.4 Sphere

A função Sphere é representada pela figura 3.3 e é caracterizada por $f(x_1 \cdots x_n) = \sum_{i=1}^{n} x_i^2$ sujeito a $-5.12 \leq x_i \leq 5.12$ com mínimo global alocado em $f(0, \cdots, 0) = 0$.

3.0.5 Rastrigin

A função Rastrigin é representada pela figura 3.4 e é caracterizada por $f(x_1 \cdots x_n) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$ sujeito a $-5.12 \leq x_i \leq 5.12$ com mínimo global alocado em $f(0, \cdots, 0) = 0$.

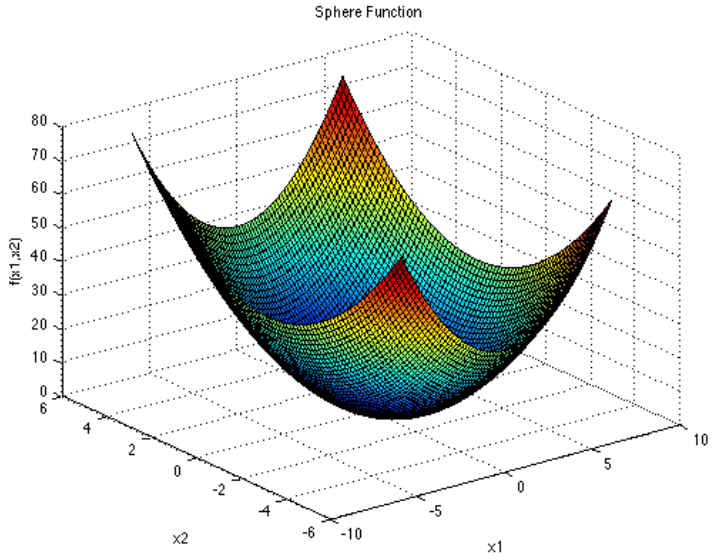

Figura 3.3 - Função Sphere

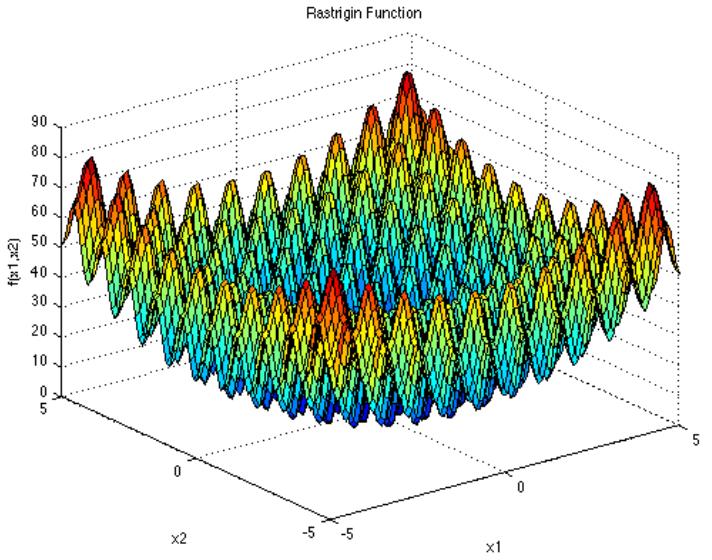

Figura 3.4 - Função Rastrigin

3.0.6 Griewank

A função Griewank é representada pela figura 3.5 e é caracterizada por $f(x_1 \cdots x_n) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$ sujeito a $-512 \leq x_i \leq 512$ com mínimo global alocado em $f(0, \cdots, 0) = 0$.

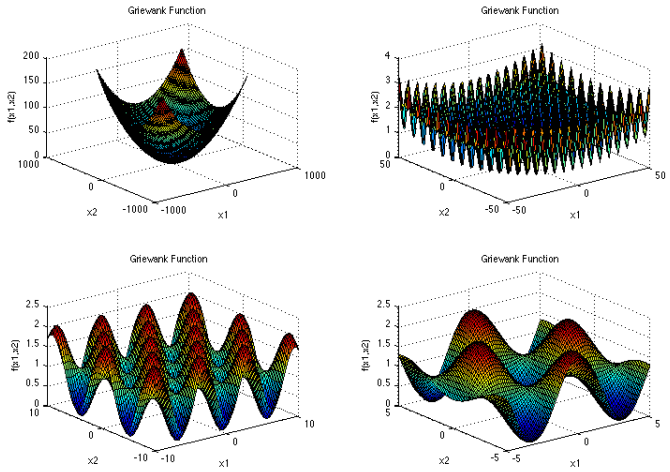

Figura 3.5 - Função Griewank

3.0.7 Ackley 1

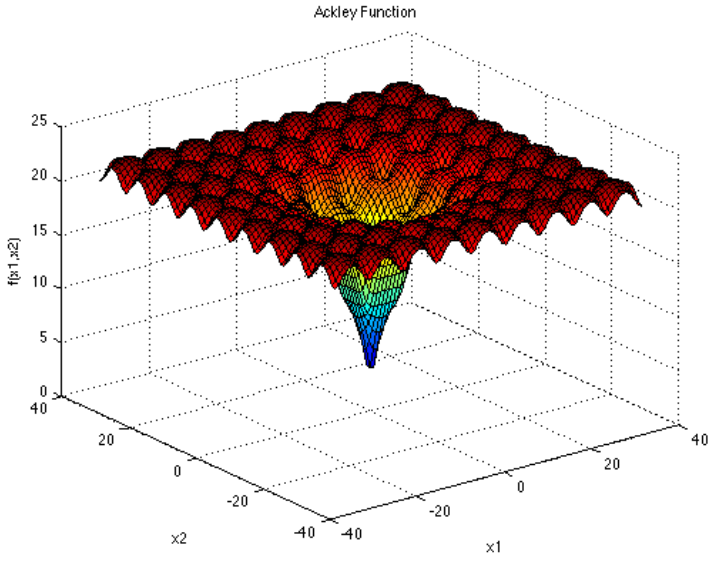

Figura 3.6 - Função Ackley 1

A função Griewank é representada pela figura 3.6 e é caracterizada por $f(x_0 \cdots x_n) = -20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$ sujeito a $-35 \leq x_i \leq 35$ com mínimo global alocado em $f(0, \cdots, 0) = 0$.

CAPÍTULO 4

RESULTADOS

Neste capítulo é demonstrado a maneira com que o trabalho foi desenvolvido, configurado e os resultados.

4.1 Algoritmos

Após a revisão bibliográfica os algoritmos que seriam utilizados para a realização desse trabalho foram definidos. A próxima etapa foi o desenvolvimento desses algoritmos nas normas do framework jMetal.

Como citado no capítulo anterior, a versão do jMetal utilizada foi a versão 5.1. Portanto, todos os algoritmos tiveram que ser adequados a esse padrão.

O número de dimensões configuradas para os experimentos foi de 500 e a quantidade de iterações de todos os algoritmos foi de 4000. O estudo foi realizado considerando 50, 100 e 200 partículas, prezando diferentes complexidades. Todos os algoritmos foram executados de forma independente por 30 vezes com o objetivo de conseguir dados estatísticos mais confiáveis. Os parâmetros utilizados neste experimento são mostrados na tabela 4.1.

Tabela 4.1 - Valores dos Parâmetros dos Algoritmos

Algoritmo	Parâmetro	Valor	Definição
CCPSO2	$pFactor$	0.5	Valor de Probabilidade
LSSPSO	w_{Max}	0.9	Inércia Máxima
	w_{Min}	0.1	Inércia Mínima
	$c1$	1.494	Confiança em Si
	$c2$	1.494	Confiança no Enxame
	pr	0.8	Valor de Probabilidade
	r	0.1	Constante Pequena
OBL-CPSO	phi	0.1	Parâmetro Manual
SPSO	w	0.721	Inércia
	c	1.193	Centro de Gravidade Constante
VCPSO	m	25	Mínimo Intervalo de cada Segmento

O equipamento utilizado para executar todos os testes foi um Intel (R) Core Processador i7-4790K CPU 4.00GHz, 4 Núcleos, 8 processadores lógicos, 16GB de memória RAM com o sistema operacional Windows 10 Pro 64-bit. A versão do software Eclipse foi a Neon. 1a Release (4.6.1) com o Framework jMetal 5.1.

4.2 Benchmarks

A confiabilidade, eficiência e validação dos algoritmos de otimização são mensuradas quando são repetidamente executados usando um conjunto de padrões de referência ou funções de teste na literatura (JAMIL; YANG, 2013), ou seja, a qualidade dos algoritmos de otimização é avaliada usando testes de benchmarks mais comumente utilizados nos trabalhos acadêmicos. As funções de referência selecionadas para a comparação dos algoritmos nessa dissertação são apresentadas na tabela 4.2 com informações sobre suas modalidades, se são separáveis ou não separáveis e o domínio do espaço de busca. O mínimo de todas as funções é zero.

Tabela 4.2 - Funções de Benchmark

Funções	Modalidade	Separabilidade	Domínio
Schwefel 1.2	Unimodal	Não Separável	$[-100; 100]^d$
Rosenbrock	Unimodal	Não Separável	$[-30; 30]^d$
Sphere	Multimodal	Separável	$[0; 10]^d$
Rastringin	Multimodal	Não Separável	$[-5.12; 5.12]^d$
Griewank	Multimodal	Não Separável	$[-100; 100]^d$
Ackley 1	Multimodal	Não Separável	$[-35; 35]^d$

Quando o objetivo é realizar comparações entre algoritmos de otimização, essas funções são utilizadas com frequência na literatura [2], [3], [7], [21] - [24]. A modalidade de uma função está relacionada ao número de máximos ou mínimos presentes em seu gráfico plotado. A separabilidade é uma propriedade que pode afetar a otimização de problemas, uma vez que funções com variáveis inter-relacionadas (chamadas de variáveis não separáveis) muitas vezes apresentam problemas de convergência.

Algumas dessas funções já foram desenvolvidas e estão configuradas nativamente no jMetal 5.1, porém as que não estavam ainda no framework foram desenvolvidas seguindo as regras necessárias.

4.3 Resultados

Nesta seção serão apresentados os estudos de caso realizados nesse projeto, visando comparar a atuação dos algoritmos em cada benchmark, com foco principalmente no resultado final e no tempo gasto para alcançar o ponto ótimo.

4.3.1 Primeiro Estudo de Caso

Nessa primeira etapa os resultados perante a função Schwefel 1.2 serão demonstrados. O gráfico 4.1 mostra o local ótimo encontrado por todos os algoritmos em um ambiente de 500 variáveis com 50, 100 e 200 partículas.

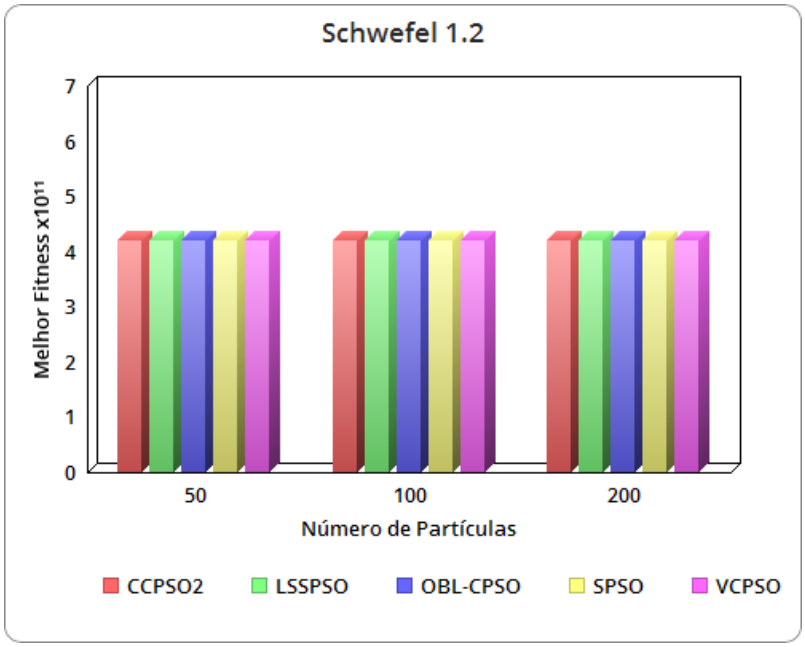

Figura 4.1 - Melhor Fitness executado na função Schwefel 1.2

Como mostrado no gráfico todos os algoritmos tiveram problemas em encontrar o ponto ótimo. Todos eles ficaram presos no ponto 4.18E+11, ou seja, em razão do estilo dessa função nenhum algoritmo foi capaz de espacar desse ponto.

Analisando o gráfico 4.2 que mostra o tempo de execução de cada algoritmo para resolver a função Schwefel 1.2, vemos que o algoritmo OBL-CPSO se mostrou eficiente no quesito velocidade em todos os casos quando se alterou o número de partículas. Podemos também destacar a eficiência do LSSPSO quando executado

com 100 partículas, esse algoritmo se mostrou bastante robusto. A metaheurística VCPSO se mostrou bastante rápida com 50 partículas.

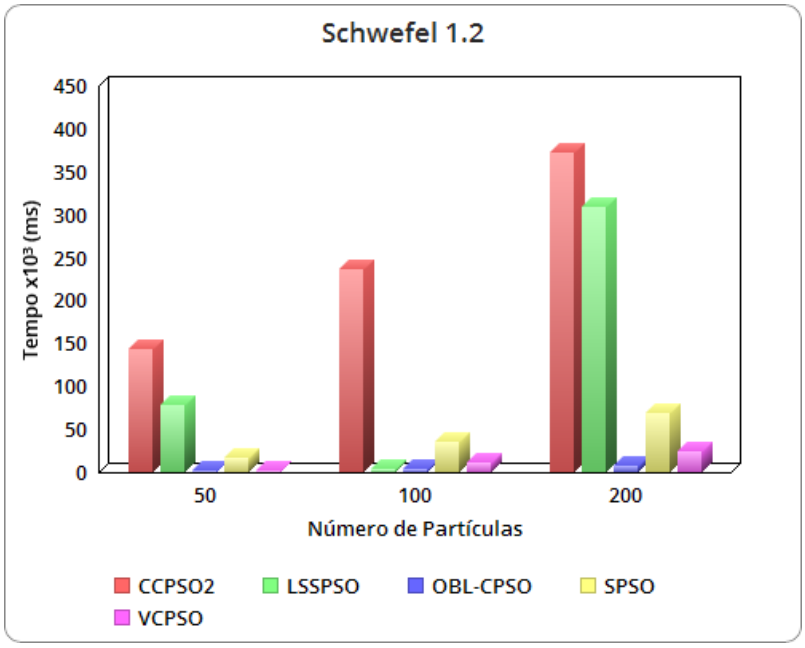

Figura 4.2 - Tempo gasto na execução na função Schwefel 1.2

4.3.2 Segundo Estudo de Caso

O segundo estudo de caso aborda os resultados sobre a função Rosenbrock. Analisando o gráfico 4.3 podemos concluir que o algoritmo que se saiu melhor foi o CCPSO2, no qual chegou a zerar o valor do fitness com 100 e 200 partículas, sendo que chegou muito próximo de zero em 50 partículas. O algoritmo LSSPSO também se saiu muito bem, pois nos três casos chegou próximo de zero. Nesse estudo podemos destacar a ineficiência do algoritmo VCPSO.

No quesito tempo de processamento como podemos ver no gráfico 4.4 a metaheurística que se destaca é o OBL-CPSO, no qual entre todos os concorrentes se saiu mais rápido. Nesse estudo de caso podemos destacar o desempenho do algoritmo VCPSO. Esse algoritmo também possuiu um baixo tempo de processamento, assim podemos concluir que sua convergência quando executado na função Rosenbrock foi muito rápida. Portanto concluímos que essa convergência acelerada ocasionou o baixo desempenho no valor $fitness$. O CCPSO2 obteve um ótimo resultado no valor

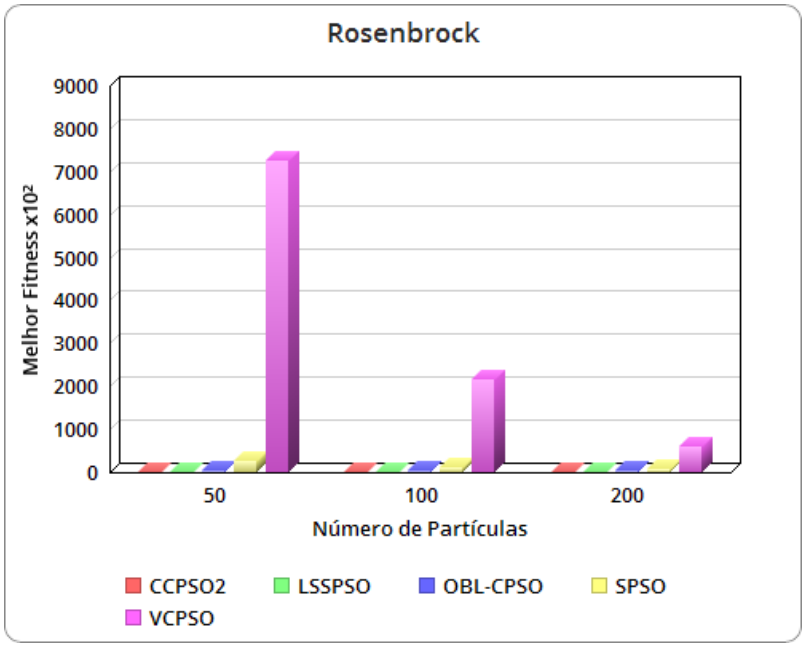

Figura 4.3 - Melhor Fitness executado na função Rosenbrock

$fitness$ porém foi o mais lento entre todos os algoritmos.

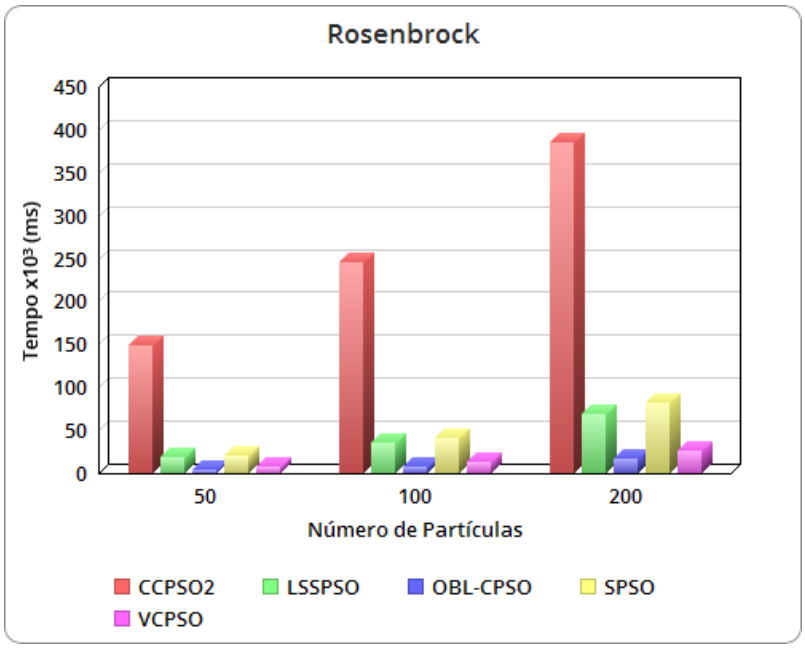

Figura 4.4 - Tempo gasto na execução na função Rosenbrock

4.3.3 Terceiro Estudo de Caso

Neste etapa analisaremos a atuação dos algoritmos quando executados para resolver a função Sphere. Como mostrado no gráfico 4.5 os algoritmos CCPSO2 e OBL-CPSO praticamente zeraram o valor $fitness$. Podemos também destacar a atuação do LSSPSO que não chegou a zerar, porém se saiu bem. Novamente o algoritmo de pior eficiência foi o VCPSO.

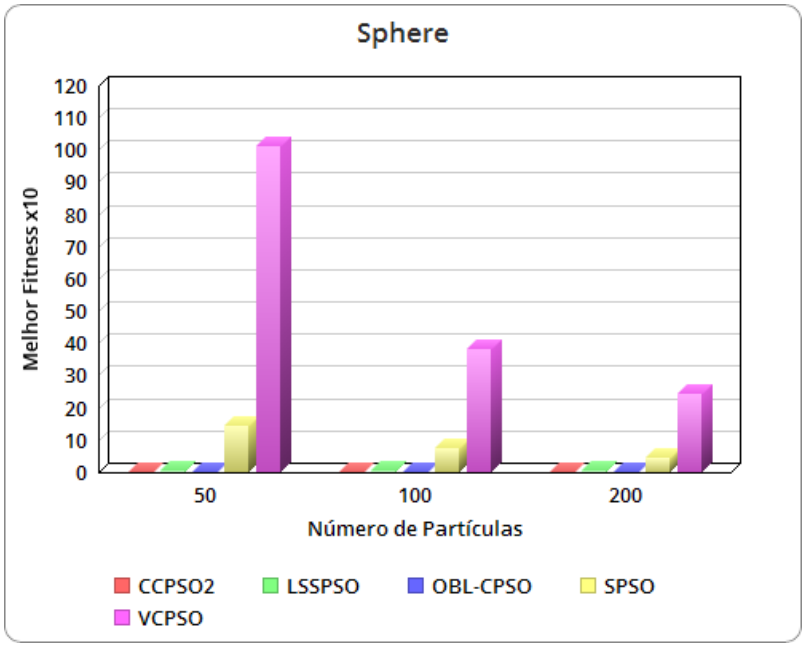

Figura 4.5 - Melhor Fitness executado na função Sphere

Analisando o gráfico 4.6 que mostra o tempo de processamento para realizar as 4000 iterações, o algoritmo OBL-CPSO se destacou positivamente. Um fato interessante no terceiro estudo de caso é que a metaheurística OBL-CPSO se destacou tanto no valor $fitness$ quanto no tempo de processamento, se mostrando um algoritmo muito robusto quando se trata de resolver a função Sphere. O algoritmo CCPSO2 se destacou negativamente no quesito tempo de execução.

4.3.4 Quarto Estudo de Caso

Nesse estudo de caso os resultados relacionados a resolução da função Rastringin serão analisados. Segundo o gráfico 4.7 podemos concluir que os algoritmos CCPSO2 e OBL-CPSO conseguiram zerar o valor do $fitness$ quando resolveram a função

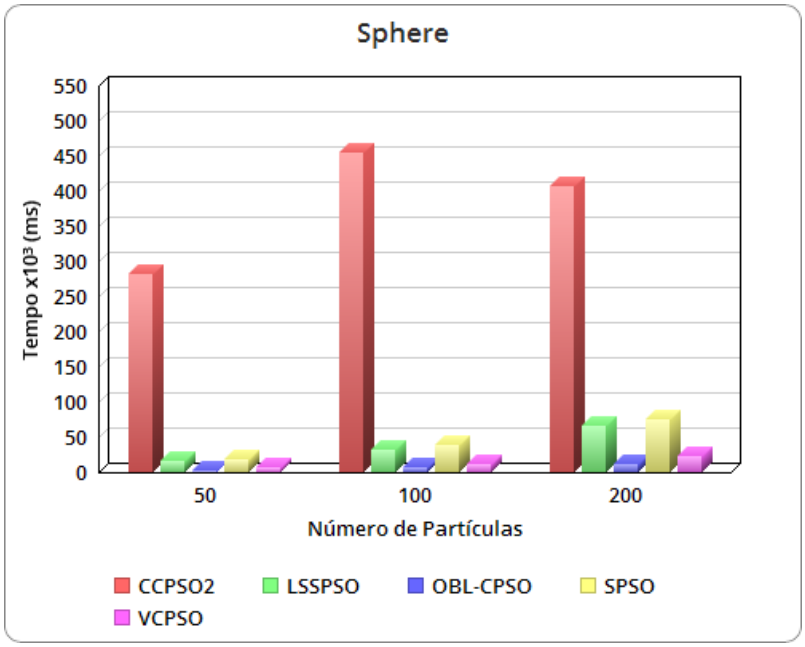

Figura 4.6 - Tempo gasto na execução na função Sphere

Rastringin. O algoritmo LSSPSO também se saiu muito bem, aproximando bastante o valor do *fitness* em zero.

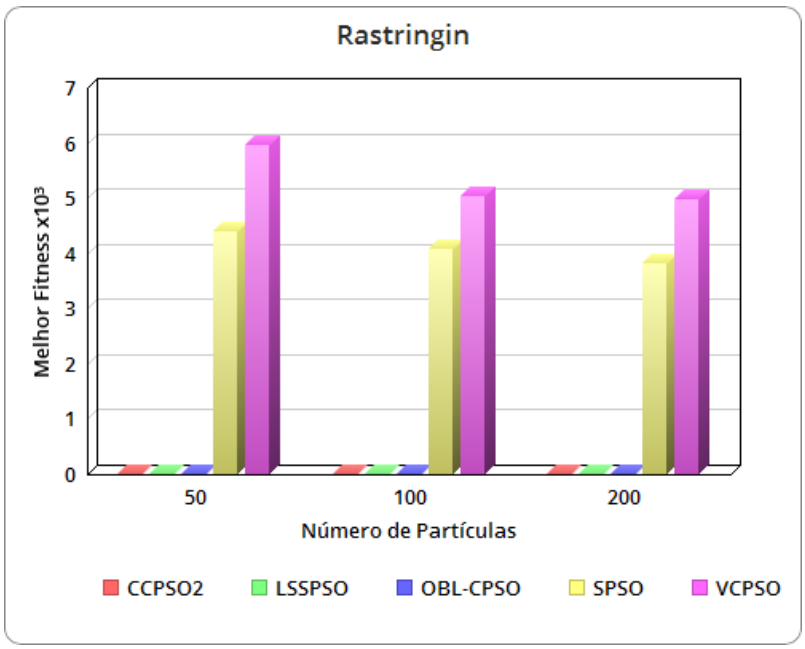

Figura 4.7 - Melhor Fitness executado na função Rastringin

Quando analisamos o tempo de execução mostrado no gráfico 4.8 concluímos que o algoritmo OBL-CPSO novamente se sai melhor que todos os seus concorrentes em todos os casos de números de partículas. Podemos destacar o bom desempenho no quesito tempo para o VCPSO no teste de 50 partículas.

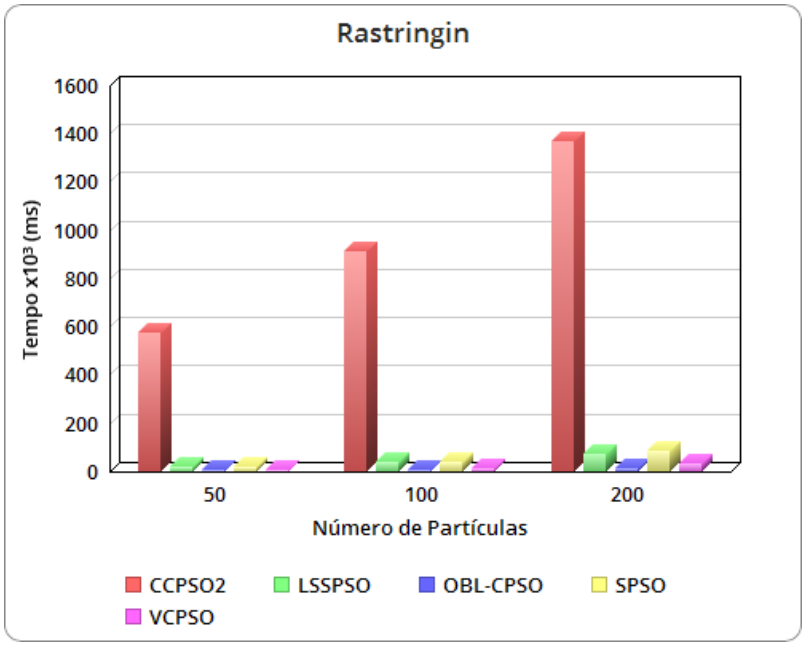

Figura 4.8 - Tempo gasto na execução na função Rastringin

4.3.5 Quinto Estudo de Caso

Neste quinto estudo de caso vamos estudar o desempenho dos algoritmos quando submetidos ao teste com o benchmark Griewank. O gráfico 4.9 mostra que os algoritmos CCPSO2 e OBL-CPSO conseguiram zerar o $fitness$ nas três situações de quantidade de partículas. O VCPSO se mostrou o menos eficiente nesse caso.

No quesito tempo de execução vemos através do gráfico 4.10 que o algoritmo OBL-CPSO além de zerar o $fitness$ também se destaca sendo o algoritmo mais rápido entre todos.

4.3.6 Sexto Estudo de Caso

Neste último estudo de caso vamos analisar o desempenho dos cinco algoritmos quando testados na função Ackley 1. Segundo o gráfico 4.11 os algoritmos CCPSO2 e

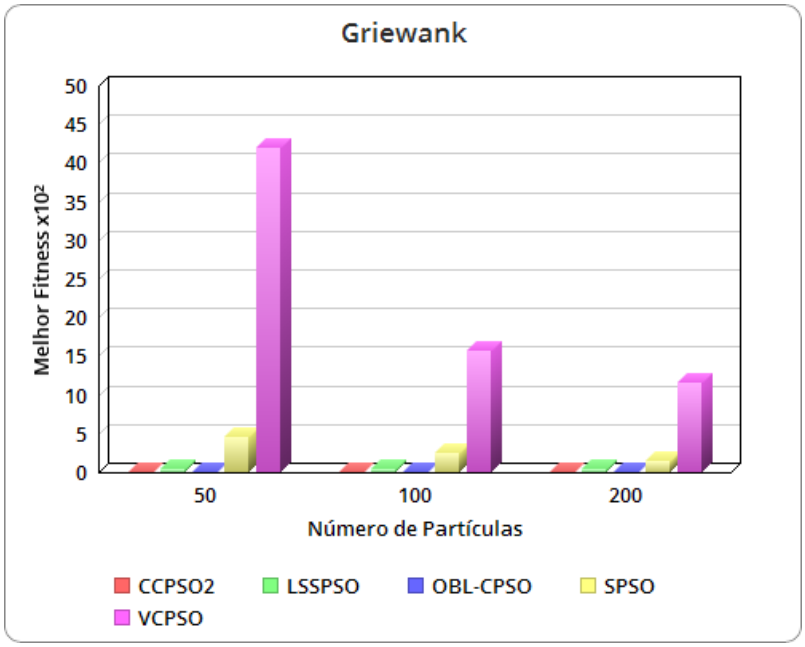

Figura 4.9 - Melhor Fitness executado na função Griewank

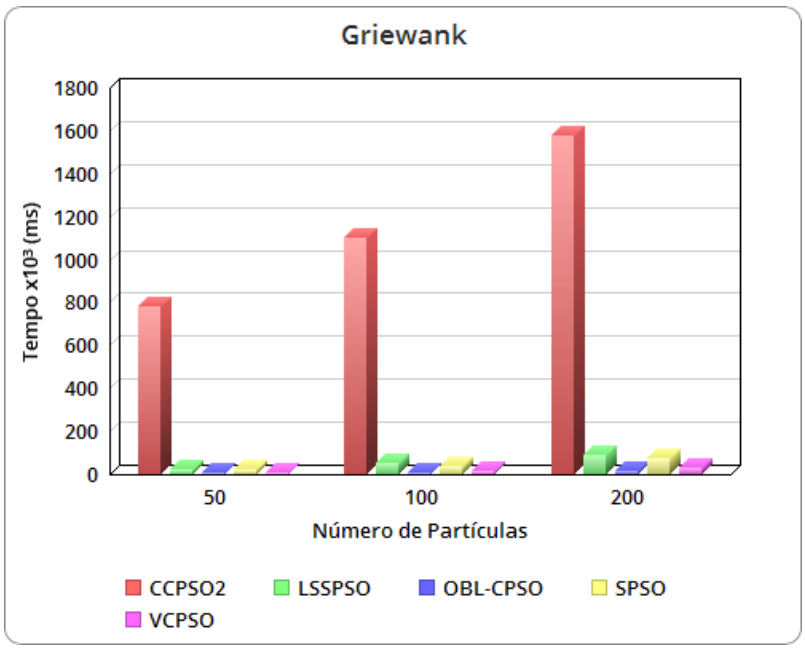

Figura 4.10 - Tempo gasto na execução na função Griewank

OBL-CPSO novamente se destacaram entre os outros. Os dois algoritmos conseguiram se aproximar bastante do valor zero. O algoritmo SPSO obteve o pior resultado entre seus concorrentes. Nos três casos em relação ao número de partículas o valor $fitness$

do SPSO ficou estagnado no mesmo valor.

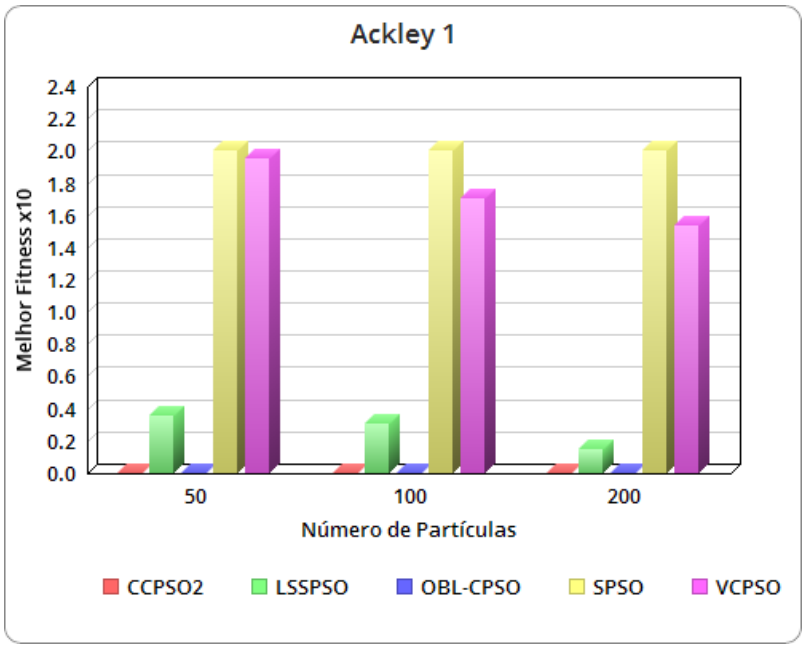

Figura 4.11 - Melhor Fitness executado na função Ackley 1

O gráfico 4.12 de tempo médio de execução mostra que o algoritmo OBL-CPSO se destacou pela velocidade de execução da função Ackley 1.

4.3.7 Resumo dos Resultados

Com a análise dos estudos de caso citados neste capítulo se faz possível tirar conclusões sobre quais algoritmos são mais eficientes para resolver problemas em larga escala. Buscando ainda mais informações para garantir a eficiência desse estudo, mais informações foram coletadas e apresentadas na tabela abaixo.

Algorithm		**CCPSO2**			**LSS-PSO**			**OBL-CPSO**			**SPSO**			**VCPSO**		
F/Particles		50	100	200	50	100	200	50	100	200	50	100	200	50	100	200
$f1$	Best	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11
	Mean	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11
	Worse	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11
	Median	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11	4,18E+11
	S.D.	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
	A. Time	1,44E+05	2,37E+05	3,72E+05	7,77E+04	**3,83E+03**	3,09E+05	**2,07E+03**	**3,83E+03**	**7,41E+03**	1,75E+04	3,46E+04	6,89E+04	**2,07E+03**	1,17E+04	2,33E+04
$f2$	Best	**6,21E-28**	**0,00E+00**	**0,00E+00**	7,75E-10	2,17E-08	1,93E-12	4,95E+02	4,95E+02	4,95E+02	2,62E+04	1,14E+04	6,04E+03	7,24E+05	2,15E+05	5,95E+04
	Mean	3,47E+02	3,24E+02	1,82E+02	3,82E-06	1,02E-05	5,68E-06	4,96E+02	4,95E+02	4,96E+02	3,60E+04	1,49E+04	7,13E+03	4,07E+06	2,79E+06	7,48E+05
	Worse	3,60E+03	3,23E+03	5,11E+02	4,50E-05	2,03E-04	5,81E-05	4,96E+02	4,95E+02	4,97E+02	4,90E+04	2,06E+04	8,82E+03	1,58E+07	1,47E+07	3,53E+06
	Median	8,20E+01	3,44E-05	9,94E-27	1,33E-06	1,40E-06	1,52E-06	4,95E+02	4,95E+02	4,96E+02	3,53E+04	1,48E+04	7,06E+03	2,65E+06	1,34E+06	2,23E+05
	S.D.	6,65E+02	6,03E+02	2,44E+02	8,47E-06	3,69E-05	1,10E-05	4,80E-01	1,96E-01	4,52E-01	6,32E+03	2,41E+03	6,83E+02	4,01E+06	3,35E+06	9,57E+05
	A. Time	1,49E+05	2,45E+05	3,84E+05	1,79E+04	3,46E+04	6,82E+04	**3,37E+03**	**6,73E+03**	**1,69E+04**	2,05E+04	4,07E+04	8,10E+04	6,96E+03	1,31E+04	2,66E+04
$f3$	Best	1,14E-154	**1,11E-193**	**1,71E-242**	1,25E-12	1,93E-15	1,14E-11	**1,51E-175**	1,09E-168	2,09E-159	1,44E+02	7,32E+01	4,42E+01	1,01E+03	3,80E+02	2,43E+02
	Mean	3,27E-06	7,49E-03	2,61E-08	4,26E-09	5,53E-09	5,95E-09	1,77E-127	5,17E-113	1,60E-122	1,73E+02	9,86E+01	5,30E+01	2,58E+03	1,07E+03	4,16E+02
	Worse	4,93E-05	2,21E-01	7,82E-07	2,54E-08	3,84E-08	7,24E-08	5,30E-126	1,55E-111	4,38E-121	2,11E+02	1,29E+02	6,58E+01	5,99E+03	3,66E+03	2,77E+03
	Median	4,19E-41	3,14E-45	8,04E-158	2,11E-09	1,20E-09	1,63E-09	3,78E-146	5,41E-140	7,57E-140	1,71E+02	9,77E+01	5,22E+01	2,33E+03	7,60E+02	2,90E+02
	S.D.	1,07E-05	4,04E-02	1,43E-07	5,79E-09	9,73E-09	1,35E-08	9,67E-127	2,83E-112	8,01E-122	1,61E+01	1,35E+01	4,96E+00	1,23E+03	8,33E+02	4,61E+02
	A. Time	2,81E+05	4,55E+05	4,07E+05	1,70E+04	3,29E+04	6,53E+04	**3,01E+03**	**5,78E+03**	**1,21E+04**	1,92E+04	3,85E+04	7,54E+04	5,71E+03	1,10E+04	2,26E+04
$f4$	Best	**0,00E+00**	**0,00E+00**	**0,00E+00**	3,64E-12	3,55E-11	9,09E-12	**0,00E+00**	**0,00E+00**	**0,00E+00**	4,40E+03	4,08E+03	3,81E+03	5,97E+03	5,02E+03	4,98E+03
	Mean	1,68E+01	1,74E+01	6,67E-04	1,35E-06	1,11E-06	9,53E-07	0,00E+00	0,00E+00	0,00E+00	4,55E+03	4,29E+03	4,02E+03	7,71E+03	6,32E+03	5,63E+03
	Worse	4,97E+02	4,97E+02	2,00E-02	6,64E-06	7,12E-06	5,69E-06	0,00E+00	0,00E+00	0,00E+00	4,68E+03	4,52E+03	4,30E+03	1,10E+04	9,37E+03	6,70E+03
	Median	3,01E-08	1,05E-11	9,09E-13	3,01E-07	1,13E-07	2,89E-07	0,00E+00	0,00E+00	0,00E+00	4,57E+03	4,29E+03	4,03E+03	7,20E+03	5,93E+03	5,38E+03
	S.D.	9,08E+01	9,07E+01	3,65E-03	1,94E-06	1,93E-06	1,42E-06	0,00E+00	0,00E+00	0,00E+00	7,58E+01	1,03E+02	1,31E+02	1,38E+03	1,20E+03	5,46E+02
	A. Time	5,78E+05	9,13E+05	1,37E+06	1,92E+04	3,77E+04	7,35E+04	**4,18E+03**	**6,91E+03**	**1,37E+04**	2,07E+04	4,18E+04	8,31E+04	8,30E+03	1,59E+04	3,19E+04
$f5$	Best	**0,00E+00**	**0,00E+00**	**0,00E+00**	1,78E-14	1,17E-14	2,22E-16	**0,00E+00**	**0,00E+00**	**0,00E+00**	4,49E+02	2,56E+02	1,47E+02	4,20E+03	1,58E+03	1,15E+03
	Mean	5,36E-03	2,94E-06	9,23E-06	3,81E-11	1,48E-11	3,66E-11	0,00E+00	0,00E+00	0,00E+00	6,16E+02	3,24E+02	1,77E+02	9,15E+03	4,83E+03	2,62E+03
	Worse	1,02E-01	4,31E-05	2,66E-04	4,18E-10	4,46E-11	2,42E-10	0,00E+00	0,00E+00	0,00E+00	8,98E+02	3,85E+02	2,23E+02	1,79E+04	1,07E+04	7,75E+03
	Median	0,00E+00	0,00E+00	0,00E+00	7,21E-12	9,89E-12	1,63E-11	0,00E+00	0,00E+00	0,00E+00	6,17E+02	3,21E+02	1,75E+02	7,70E+03	4,62E+03	1,81E+03
	S.D.	1,97E-02	1,06E-05	4,85E-05	8,22E-11	1,42E-11	6,18E-11	0,00E+00	0,00E+00	0,00E+00	1,13E+02	3,28E+01	1,66E+01	4,33E+03	2,38E+03	1,78E+03
	A. Time	7,78E+05	1,10E+06	1,58E+06	2,60E+04	5,02E+04	8,64E+04	**4,20E+03**	**8,24E+03**	**1,66E+04**	1,89E+04	3,79E+04	7,58E+04	8,41E+03	1,61E+04	3,19E+04
$f6$	Best	7,55E-15	3,24E-14	7,55E-15	3,60E+00	3,12E+00	1,47E+00	**4,44E-16**	**4,44E-16**	**4,44E-16**	2,00E+01	2,00E+01	2,00E+01	1,95E+01	1,71E+01	1,54E+01
	Mean	7,93E-01	3,47E-04	4,04E-12	1,83E+01	1,94E+01	1,92E+01	2,00E+00	1,33E+00	1,24E-01	2,00E+01	2,00E+01	2,00E+01	2,08E+01	1,97E+01	1,80E+01
	Worse	2,00E+01	7,87E-03	1,06E-10	2,12E+01	2,12E+01	2,11E+01	2,00E+01	2,00E+01	3,71E+00	2,00E+01	2,00E+01	2,00E+01	2,17E+01	2,11E+01	2,13E+01
	Median	8,51E-13	4,71E-13	4,20E-13	2,11E+01	2,10E+01	2,10E+01	4,44E-16	4,44E-16	4,44E-16	2,00E+01	2,00E+01	2,00E+01	2,09E+01	2,03E+01	1,74E+01
	S.D.	3,68E+00	1,49E-03	1,93E-11	6,20E+00	4,94E+00	4,64E+00	6,10E+00	5,07E+00	6,78E-01	0,00E+00	0,00E+00	0,00E+00	6,23E-01	1,23E+00	1,72E+00
	A. Time	5,69E+05	9,29E+05	1,32E+06	8,69E+04	1,88E+05	3,75E+05	**3,12E+03**	**5,82E+03**	**1,08E+04**	1,81E+04	3,58E+04	7,29E+04	5,69E+05	1,77E+04	3,60E+04

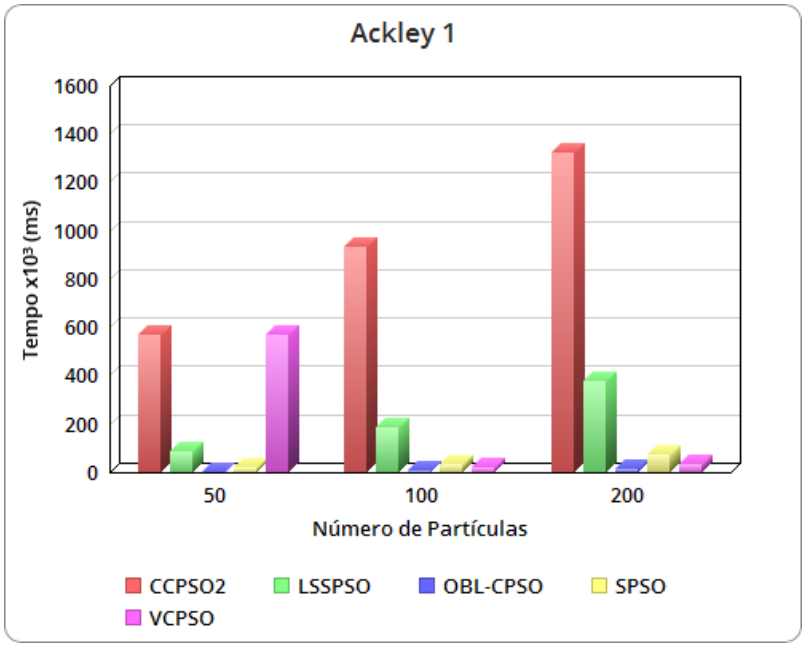

Figura 4.12 - Tempo gasto na execução na função Ackley 1

CAPÍTULO 5

CONCLUSÃO

Neste projeto experimentos foram realizados visando comparar cinco algoritmos quando submetidos a otimização de problemas em larga escala. Os experimentos foram realizados utilizando seis funções de benchmark muito usadas na literatura quando se trata de comparação de algoritmos de otimização.

Visando a resolução de problemas reais que em sua maioria são de larga escala, os estudos mostraram que os algoritmos CCPSO2, LSSPSO e OBL-CPSO possuem uma boa performance na resolução de problemas em larga escala.

Após todos os experimentos podemos concluir que o algoritmo CCPSO2 demonstrou que é capaz de resolver praticamente a maioria dos problemas de larga escala, porém sua média de tempo gasto para finalizar a execução é muito grande. Portanto, dependendo do caso a ser otimizado o CCPSO2 pode ser muito útil, desde que esse problema permita que o resultado demore um tempo maior para ser alcançado.

O algoritmo OBL-CPSO se mostrou bastante eficiente em muitos experimentos e estudos de caso. Um importante dado é que esse algoritmo atingiu o ótimo global em dois benchmarks em todas as execuções. Portanto, o OBL-CPSO pode ser considerado um algoritmo poderoso capaz de resolver problemas complexos do mundo real de forma mais rápida que o algoritmo CCPSO2.

Outro algoritmo a se destacar é o LSSPSO, uma vez que ele obteve alguns bons resultados e se mostrou um algoritmo bem estável.

5.1 Contribuições do Trabalho

Artigos em congresso: O estudo dessa dissertação possibilitou a publicação do artigo Comparison of PSO Variants Applied to Large Scale Optimization Problems no LA-CCI 2017 - IEEE Latin American Conference on Computational Intelligence. (ULLMANN et al., 2017)

5.2 Sugestões para Trabalhos Futuros

Como sugestão de trabalhos futuros os algoritmos podem ser submetidos a outros benckmarks e também podem ser testados em dimensões maiores e menores. Outra sugestão importante é a utilização dos algoritmos já testados e que obtiveram bons resultados (CCPSO2, OBL-CPSO e LSSPSO) na resolução de problemas em larga escala de sistemas hidrotérmicos.

REFERÊNCIAS BIBLIOGRÁFICAS

BRATTON, D.; KENNEDY, J. Defining a standard for particle swarm optimization. **2007 IEEE Swarm Intelligence Symposium**, 2007. 14

CHENG, R.; JIN, Y. A competitive swarm optimizer for large scale optimization. **IEEE transactions on cybernetics**, v. 45, n. 2, p. 191–204, 2014. 30

CICOGNA, M. Sistema de suporte a decisao para o planejamento e a programacao da operacao de sistemas de energia eletrica. **Sistema de Suporte a Decisao para o Planejamento e a Programacao da Operacao de Sistemas de Energia Eletrica**, 2003. 16

CIGOGNA, M. A.; SOARES, S. Modelo de planejamento da operacao energetica de sistemas hidrotermicos a usinas individualizadas orientado por objetos. **Universidade Estadual de Campinas, Faculdade de Engenharia Eletrica e Computacao**, 1999. 16

DING J. LIU, K. R. C. W. Z. Q. H. J.; LEI, J. A particle swarm optimization using local stochastic search and enhancing diversity for continuous optimization. **Neurocomputing**, v. 137, p. 261–267, 2014. 14, 26

DURILLO, J. J.; NEBRO, A. J. jmetal: A java framework for multi-objective optimization. **Advances in Engineering Software**, v. 42, n. 10, p. 760–771, 2011. 14, 42

JAMIL, M.; YANG, X. S. A literature survey of benchmark functions for global optimisation problems. **International Journal of Mathematical Modelling and Numerical Optimization**, v. 4, n. 2, p. 150, 2013. 14, 43, 48

KENNEDY, J.; EBERHART, R. Particle swarm optimization. **IEEE International of first Conference on Neural Networks**, 1995. 13, 14

KENNEDY, R. E. J. F.; SHI, Y. Swarm intelligence. **Morgan Kaufmann**, 2001. 15

LI, X.; YAO, X. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. **2009 IEEE Congress on Evolutionary Computation**, 2009. 23

_____. Cooperatively coevolving particle swarms for large scale optimization. **IEEE Transactions on Evolutionary Computation**, v. 16, n. 2, p. 210–224, 2012. 14

M. ELIZABETH F. W., L. M. C. P. A. V. M. C. G. Aplicacao da metaheuristica deepso a problemas de otimizacao global em larga escala. **XLVII SBPO**, p. 1758–1767, 2015. 17

MARQUES, T. Uma politica operativa a usinas individualizadas para o planejamento da operacao energetica do sistema interligado nacional. **Tese (Doutorado) - Universidade Estadual de Campinas**, 2006. 16

MOLGA, M.; SMUTNICKI, C. Test functions for optimization needs. **Test functions for optimization needs**, 2005. 14, 43

NEBRO, A. J.; DURILLO, J. J.; VERGNE, M. Redesigning the jmetal multi-objective optimization framework. **Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion 15**, 2015. 14, 42

RAHNAMAYAN, S.; TIZHOOSH, H.; SALAMA, M. Opposition-based differential evolution. **IEEE Transactions on Evolutionary Computation**, v. 12, n. 1, p. 64–79, 2008. 30

TIZHOOSH, H. Opposition-based learning: A new scheme for machine intelligence. **International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC06)**, 2005. 30

ULLMANN, M. R. D.; PIMENTEL, K. F.; MELO, L. A. D.; CRUZ, G. D.; VINHAL, C. Comparison of pso variants applied to large scale optimization problems. **2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)**, 2017. 59

WILSON, E. Sociobiology: The new synthesis. **Belknap Press**, 1975. 19

ZHANG, Q.; LIU, W.; MENG, X.; YANG, B.; VASILAKOS, A. V. Vector coevolving particle swarm optimization algorithm. **Information Sciences**, v. 394-395, p. 273–298, 2017. 10, 14, 36, 38

ZHOU W. FANG, X. W. J. S. J.; CHENG, S. An opposition-based learning competitive particle swarm optimizer. **Evolutionary Computation (CEC), 2016 IEEE Congress on. IEEE**, 2016. 10, 14, 31