UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniel Leal Souza

Otimização por Multi-Enxame Evolucionário de Partículas Clássico e Quântico Competitivo sob a Arquitetura Paralela CUDA Aplicado em Problemas de Engenharia

Belém
2014

Daniel Leal Souza

Otimização por Multi-Enxame Evolucionário de Partículas Clássico e Quântico Competitivo sob a Arquitetura Paralela CUDA Aplicado em Problemas de Engenharia

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (Área de concentração: Sistemas Inteligentes) como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Universidade Federal do Pará.

Orientador: Prof. Dr. Dionne Cavalcante Monteiro (PPGCC/UFPA)

Co-orientador: Prof. Dr. Roberto Célio Limão de Oliveira (PPGEE/UFPA)

Belém
2014

Dados Internacionais de Catalogação na Publicação (CIP)

Biblioteca Central da UFPa

Daniel Leal Souza, Autor, 2014

Otimização por Multi-Enxame Evolucionário de Partículas Clássico e Quântico Competitivo sob a Arquitetura Paralela CUDA Aplicado em Problemas de Engenharia / Orientador: Dionne Cavalcante Monteiro; Co-orientador: Roberto Célio Limão de Oliveira - 2014

Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Ciências Exatas e Naturais, Programa de Pós-Graduação em Ciência da Computaçção, Belém-Pa, 2014.

1. PSO. 2. EPSO. 3. QPSO 4. PSO+EE 5. QPSO+EE 6. COMSO 7. COEMSO 8. COQMSO 9. CEMSO 10. CQEMSO 11. GPGPU. 12. GPU. 13. CUDA. 14. NVIDIA. 15. Enxame. 16. Cooperativo. 17. Competitivo. 18. Paralelismo. 19. Computação Heterogênea.

CDD - 22. ed. 004.6

Daniel Leal Souza

Otimização por Multi-Enxame Evolucionário de Partículas Clássico e Quântico Competitivo sob a Arquitetura Paralela CUDA Aplicado em Problemas de Engenharia

Dissertação de Mestrado apresentada para obtenção do grau de Mestre em Ciencia da Computação. Programa de Pós-Graduação em Ciência da Computação (área de concentração: Sistemas Inteligentes). Instituto de CiÊncias Exatas e Naturais. Universidade Federal do Pará.

Banca Examinadora:

Prof. Dr. Dionne Cavalcante Monteiro
Programa de Pós-Graduação em Ciência da Computação - UFPA - Orientador

Prof. Dr. Roberto Célio Limão de Oliveira
Programa de Pós-Graduação em Engenharia Elétrica - UFPA - Co-orientador

Prof. Dr. Alexandre Cláudio Botazzo Delbem
Instituto de Ciências Matemáticas e de Computação - USP - Membro

Prof. Dr. Claudemiro Sales
Programa de Pós-Graduação em Ciência da Computação - UFPA - Orientador

---

Prof. Dr. Nelson Cruz Sampaio
(COORDENADOR DO PPGCC/ICEN - UFPA)

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS - ICEN
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Otimização por Multi-Enxame Evolucionário de Partículas Clássico e Quântico sob a Arquitetura Paralela CUDA Aplicado em Problemas de Engenharia

Daniel Leal Souza

Esta Dissertação foi julgada adequada para obtenção do título de Mestre em Ciência da Computação com ênfase em Sistemas Inteligentes e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Banca examinadora:

Prof. Dr. Dionne Cavalcante Monteiro
(ORIENTADOR - UFPA)

Prof. Dr. Roberto Célio Limão de Oliveira
(CO-ORIENTADOR - UFPA)

Prof. Dr. Alexandre Cláudio Botazzo Delbem
(MEMBRO - ICMC/USP)

Prof. Dr. Claudemiro Sales
(MEMBRO - UFPA)

Visto

Prof. Dr. Nelson Cruz Sampaio
(COORDENADOR DO PPGCC/ICEN - UFPA)

Belém, Setembro de 2014

ESTE TRABALHO É DEDICADO AOS MUITOS JOVENS, HOMENS E MULHERES DESTE PAÍS QUE NÃO TIVERAM A OPORTUNIDADE DE UMA EDUCAÇÃO DIGNA E DE QUALIDADE DEVIDO O DESCASO E A FALTA DE INTERESSE DE POUCOS...

APENAS A EDUCAÇÃO PODE MUDAR ESTE PAÍS!

"Deixo-vos a paz, a minha paz vos dou; não vo-la dou como o mundo a dá. Não se turbe o vosso coração, nem se atemorize.".

Jesus Cristo (João 14:27)

”O Amor é o Meio, a Verdade é o Fim... Se chegarmos ao Meio, cedo ou tarde chegaremos ao Fim, a Verdade, a Deus” .

Mahatma Gandhi

”Sim! Grandes coisas fez o SENHOR por nós, por isso estamos alegres”.

Salmos 126:3

"Ele (Deus) é o dono de tudo. Devo a Ele a oportunidade que tive de chegar onde cheguei. Muitas pessoas têm essa capacidade, mas não têm a oportunidade. Ele a deu para mim, não sei porque. Só sei que não posso desperdiça-la".

Ayrton Senna

”Deus é grande, Deus é forte, Deus é justo! Quando Ele quer, não tem quem não queira”.

Ayrton Senna

Agradecimentos

Acima de tudo, à **Deus** pela oportunidade de viver em uma família onde **valores, princípios, integridade, fé, amor e dignidade** são cultivados. Tudo que sou hoje devo a família maravilhosa e amorosa que Ele me proporcionou: À minha mãe *Solange Maria Leal Souza*, meu pai *Pedro Valdo Saldanha Souza* e meu irmão *Lucas Leal Souza*, dedico minha eterna gratidão por tudo que fizeram, fazem e hão de fazer por mim.

Ao professor, orientador e grande amigo Dr. *Dionne Cavalcante Monteiro* pelos ensinamentos, sabedoria, direcionamento e apoio irrestrito prestado. Deixo ao Sr. minha eterna gratidão... As lembranças dos momentos que fizeram parte desta jornada levarei para o resto da minha vida!

Ao professor, co-orientador e amigo Dr. *Otávio Noura Teixeira* pelas oportunidades concedidas, motivação, amizade e confiança que contribuíram diretamente para minha formação profissional. Muito obrigado!

Ao professor, co-orientador e amigo Dr. *Roberto Célio Limão de Oliveira* pelos conselhos e direcionamento, fundamentais para o desenvolvimento deste trabalho. Muito obrigado!

Aos grandes amigos de longa data *Elizabete Oliveira* e *Prof. Dr. Tiago Carvalho Martins* (UFPA) por acreditarem no meu potencial. Agradeço de coração pela oportunidade que vocês me apresentaram, sem a qual não seria possível me tornar o profissional que hoje sou. Deixo a vocês minha eterna gratidão.

Ao professor e amigo Dr. *Victor Alexandrovich Dmitriev* (UFPA) pela oportunidade oferecida, podendo assim desenvolver minhas habilidades como pesquisador, fundamentais para a minha formação. Serei sempre grato pela confiança depositada em mim!

Aos amigos *Glauber Duarte Monteiro*, *Prof. MSc. Renato Hidaka*, *Walter Avelino*, *Roberto Franco*, *Marco Mollinetti*, Prof. Dr. *Claudomiro Sales*, casal *Cristina e Brent Wilberg*, casal *Jussie Gonçalves e Waldelice de Souza* e a todos que de alguma forma, contribuíram para que esse trabalho fosse possível.

Agradeço a Fundação de Amparo à Pesquisa do Estado do Pará pelo apoio financeiro prestado durante o peródo do mestrado.

Por fim, à equipe do Laboratório de Computação Natural (LCN-CESUPA) e do Programa de Pós-Graduação em Ciência da Computação (PPGCC-UFPA) pelo apoio e contribuição neste trabalho e que ajudaram, direta ou indiretamente, na realização deste sonho.

Deixo um agradecimento especial ao professor e amigo *Dr. Sandro Bezerra* (PPGCC-UFPA). Sua ajuda, direcionamento e crença no meu potencial foram imprescindíveis para a realização desta etapa da minha vida. **SEREI ETERNAMENTE GRATO PELA CONFIANÇA DEPOSITADA. QUE DEUS LHE ABENÇOE!**

À TODOS VOCÊS, MUITO OBRIGADO!

Daniel Leal Souza

In Memoriam...

Este trabalho é dedicado a memória dos estudantes de Ciência e Engenharia da Computação, amigos e colegas da UFPA e demais instituições que perderam suas vidas em 16/07/2012, durante uma excursão ao Congresso da Sociedade Brasileira de Computação (CSBC 2012) em Curitiba (PR).

Ariane dos Santos Sinimbu

Bruno Azevedo Amaral

Darlyce de Lima Cavalcante

Felipe José Barros Amim

Ivana Dias Rosa

Leonardo Lopes Couto

Renato Iori da Conceição

Sivaldo Miranda do Nascimento Junior

Thamirys dos Santos Oliveira

Wilson Louzeiro Evangelista

"Nunca digo adeus a ninguém.
Nunca deixo que as pessoas
mais próximas a mim se vão.
Levo-as comigo aonde vou."

Autor Desconhecido

Resumo

Este trabalho apresenta o desenvolvimento de um conjunto de metaheurística híbridas, baseadas na utilização das estratégias evolutivas em conjunto com os algoritmos de otimização por enxame de partículas clássica e quântica sob um ambiente multi-enxame com topologia mestre-escravos. Tais algoritmos são denominados **Competitive Evolutionary Multi-Swarm Optimization** (CEMSO) [1] e **Competitive Quantum-Behaviour Evolutionary Multi-Swarm Optimization** (CQEMSO) [2]. Para efeito de comparação e validação dos resultados, são utilizados quatro problemas de engenharia presentes em diversas publicações científicas: Projeto de Viga de Aço (WBD); Peso da Tensão/Compressão sobre Mola (MWTCS); Projeto de Redutor de Velocidade (SRD); Projeto de Vaso de Pressão (DPV).

Em relação a implementação, os algoritmos foram desenvolvidos sob a arquitetura CUDA, a qual proporciona um ambiente de computação paralela massiva que viabiliza uma distribuição de dados mais adequada em relação a organização dos enxames, além de contribuir para a diminuição significativa do tempo de processamento.

Com a aplicação das estratégias evolutivas nos algoritmos PSO e QPSO, bem como os mecanismos de condições de contorno propostos, as soluções descritas neste documento oferecem diversas vantagens, onde se pode destacar melhorias na capacidade de busca, aumento na taxa de convergência e alto grau de paralelismo. Tais fatos são confirmados através dos dados obtidos (i.e. Tempo de execução, melhores soluções obtidas, média e variância de resultados) pelos algoritmos CEMSO e CQEMSO em relação as versões multi-enxame dos algorimos PSO (COMSO), EPSO (COEMSO) e COQMSO (QPSO), todos implementados e submetidos a análise de desempenho através dos experimentos com problemas de engenharia.

Palavras-chave: PSO, EPSO, QPSO, PSO+EE, QPSO+EE, COMSO, COEMSO, COQMSO, CEMSO, CQEMSO, GPGPU, CUDA, Enxame, Paralelismo.

[1] Em português: Otimização por Multi-Enxame Evolucionário de Partículas Competitivo

[2] Em português: Otimização por Multi-Enxame Evolucionário de Partículas Competitivo com Inspiração Quântica

Abstract

This paper presents the development of a set of hybrid metaheuristic based on the use of evolutionary strategies in conjunction with classical and quantum multi-swarm optimization with master-slave approach. These algorithms are named **Competitive Evolutionary Multi-Swarm Optimization** (CEMSO) and **Competitive Quantum-Behaviour Evolutionary Multi-Swarm Optimization** (CQEMSO). For comparison and validation of the results, four engineering problems encountered in many publications scientific are used: Welded Beam Design (WBD); Minimization of the Weight of a Tension/ Compression Spring (MWTCS); Speed Reducer Design (SRD); Design of a Pressure Vessel (DPV).

The algorithms were developed under the CUDA architecture, which provides a massive parallel computing environment that enables a more appropriate data allocation regarding the organization of swarms, as well as contributing to the significant decrease in processing time.

With the application of evolutionary strategies in the PSO and QPSO algorithms, as well as the proposed boundary conditions, the solutions described in this document offer several advantages. We can highlight improvements in the ability to search, increasing the convergence rate and high degree of parallelism. These facts are confirmed by the data obtained (i.e. Execution time, best solutions obtained, mean and variance of results) by CQEMSO and CQEMSO algorithms when compared to those obtained from multi-swarm approach for PSO (COMSO), EPSO (COEMSO) and QPSO (COQMSO). All of these algorithms were implemented and subjected to performance analysis through experiments with engineering problems described above.

Keywords: PSO, EPSO, QPSO, PSO+EE, QPSO+EE, COMSO, COEMSO, COQMSO, CEMSO, CQEMSO, GPGPU, CUDA, Swarm, Parallelism.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Publicações

No decorrer do curso de mestrado, foram elaboradas duas publicações em conferências e uma para capítulo de livro. Estas publicações estão ordenadas em ordem cronológica, revelando a evolução gradual dos estudos até consolidação da proposta sugerida neste trabalho.

Publicações em congresso

1- Souza, D. L.; Teixeira, O. N.; Monteiro, D. C.; Oliveira, R. C. L. *A New Cooperative Evolutionary Multi-Swarm Optimizer Algorithm Based on CUDA Parallel Architecture Applied to Solve Engineering Optimization Problems*. ECAI 2012: European Congress on Artificial Inteligence (Workshop CIMA 2012: 3rd International Workshop on Combinations of Intelligent Methods and Applications), Montpelier - França (2012)

2- Souza, D. L.; Teixeira, O. N.; Monteiro, D. C.; Oliveira, R. C. L.; Mollinetti, M. A. F. *A Novel Competitive Quantum-Behaviour Evolutionary Multi-Swarm Optimizer Algorithm Based on CUDA Architecture Applied to Constrained Engineering Design*. ANTS 2014: 9th International Conference on Swarm Intelligence, Bruxelas - Bélgica (2014)

Publicações em capítulo de livro (*Proceedings*)

3- Souza, D. L.; Monteiro, G. D.; Teixeira, O. N.; Martins, T. C.; Dmitriev, V. A. *PSO-GPU: Accelerating Particle Swarm Optimization in CUDA-Based Graphics Processing Units*. GECCO '11: Proceedings of The 13th Annual Conference Companion on Genetic and Evolutionary Computation, 2011, Dublin - Irlanda (2011), Páginas: 837-838, ISBN: 978-1-4503-0690-4, DOI:10.1145/2001858.2002114

4- Souza, D. L.; Teixeira, O. N.; Monteiro, D. C.; Oliveira, R. C. L. *A New Cooperative Evolutionary Multi-Swarm Optimizer Algorithm Based on CUDA Architecture Applied to Engineering Optimization*. Combinations of Intelligent Methods and Applications (CIMA 2012), Springer-Veerlag (2013), Págins: 95-115, ISBN: 978-3-642-36651-2

5- Souza, D. L.; Teixeira, O. N.; Monteiro, D. C.; Oliveira, R. C. L.; Mollinetti, M. A. F. *A Novel Competitive Quantum-Behaviour Evolutionary Multi-Swarm Optimizer Algorithm Based on CUDA Architecture Applied to Constrained Engineering Design*. Swarm Intelligence - Proceedings of the 9th International Conference on Swarm Intelligence (ANTS 2014), Springer-Veerlag (2014), Intervalo de página e ISBN a serem publicados

Lista de Símbolos e Abreviaturas

AMBER: Assisted Model Building with Energy Refinement

API: *Application Programming Interface* ou interface de programação de aplicativos

APIC: *Advanced Programmable Interrupt Controller* ou Controlador Avançado de Interrupção Programável

ASMP: *Asymmetric Multiprocessing*, ou processamento assimétrico

ATD: Air Tank Design

CEMSO: Multi-Enxame Evolucionário Cooperativo de Partículas

CERN: *Conseil Européen pour la Recherche Nucléaire*, ou Organização Europeia para a Pesquisa Nuclear

CPTEC: Centro de Previsão de Tempo e Estudos Climáticos

CPU: *Central Processing Unit* ou unidade de processamento central

CQEMSO: Multi-Enxame Evolucionário Cooperativo de Partículas com Inspiraçõ Quântica

CUBLAS: *CUDA Basic Linear Algebra System*

CUDA: *Computer Unified Device Architecture*. Arquitetura para desenvolvimento de aplicaçõs de propósito geral para GPUs

CUFFT: *CUDA Fast Fourrier Transformation*

DirectX: API para desenvolvimento de aplicativos gráficos da Microsoft

DMMP: *Distributed Memory Message Passing*

DMSV: *Distributed Memory Shared Variable*

DPV: Design of a Pressure Vessel

DRAM: *Dynamic Random Access Memory*. Memória RAM de acesso direto que armazena cada bit de dados num condensador ou capacitor

EIB: *Element Interconect Bus*

EPSO: *Evolutionary Particle Swarm Optimization* ou otimização por enxame de partículas evolucionário

FPGA: *Field Programmable Gate Array Architecture*

GCC: *GNU C Compiler*

GCD: *Grand Central Dispatch*

GMMP: *Global Memory Message Passing*

GMSV: *Global Memory Shared Variable*

GPGPU: *General-Purpose Computing on Graphics Processing Units* ou computação de propósito geral em unidades de processamento gráfico

GPU: *Graphics Processing Unit* ou unidade de processamento gráfico

IEEE: Instituto de Engenheiros Eletricistas e Eletrônicos

INPE: Instituto Nacional de Pesquisas Espaciais

LHC: *Large Hadron Collider*, ou Grande Colisor de Hadrons

MIMD: *Multiple Instruction, Multiple Data*

MISD: *Multiple Instruction, Single Data*

MPI: *Message Passing Interface*

MSIMD: *Multiple Single Instruction, Multiple Data*

MWTCS: Minimization of the Weight of a Tension/ Compression Spring

NVCC: *NVIDIA C Compiler*

OpenACC: API para programação paralela baseada em diretivas

OpenCL: *Open Computing Language*. Framework usado para desenvolver aplicações que execute operações através de arquiteturas hetergêneas (e.g. CPU, GPU)

OpenGL: *Open Graphics Library*. API para desenvolvimento de aplicativos gráficos open-source

OpenMP: *Open Multi-Processing*. API para a programação *multi-thread* de memória compartilhada

PPE: *Power Processing Element*

PSO: *Particle Swarm Optimization* ou otimização por enxame de partículas

PVM: *Parallel Virtual Machine*

QPSO: *Quantum-Behaved Particle Swarm Optimization* ou otimização por enxame de partículas com inspiração quântica

ROP: *Render Output Unit*, ou Unidade de Saída de Renderização

SIGA: Acrônimo para *Social Interaction with Genetic Algorithm*, ou Algoritmo Genético com Interação Social

SIGA: Acrônimo para *Social Interaction with Genetic Algorithm*, ou Algoritmo Genético com Interação Social

SIGA: Acrônimo para *Social Interaction with Genetic Algorithm*, ou Algoritmo Genético com Interação Social

SIMD: *Single Instruction, Multiple Data*

SIMT: *Single Instruction, Multiple Threads*

SISD: *Single Instruction, Single Data*

SM: *Streaming Multiprocessor*, modelo de multiprocessadores de fluxo encontrado nas arquiteturas Fermi ou anterior

SMP: *Symmetric Multiprocessing*, ou processamento simétrico

SMX: *Streaming Multiprocessor X*, modelo de multiprocessadores de fluxo encontrado em GPUs da arquitetura Kepler

SP: *Streaming Processor*, ou processador de fluxo

SPE: *Synergistic Processing Elements*

SPMD: *Single Program, Multiple Data*

SRD11: Speed Reducer Design for 11 Restrictions

TCD: Tubular Column Design

ULA: Unidade Lógica e Aritmética

WBD: Welded Beam Design

x86: Arquitetura de processadores baseados no processador Intel 8086

Capítulo 1

Introdução

Este capítulo apresenta os tópicos introdutórios relacionados ao desenvolvimento do trabalho, bem como os objetivos gerais e específicos, a metodologia aplicada e a estrutura geral da dissertação. Os tópicos iniciais estão divididos em duas partes (Seções 1.1 e 1.2) no intuito de abordar de maneira detalhada os principais assuntos relacionados a esse trabalho: Computação bio-inspirada e GPGPU com ênfase na abordagem do enxame de partículas.

1.1 Computação Bioinspirada

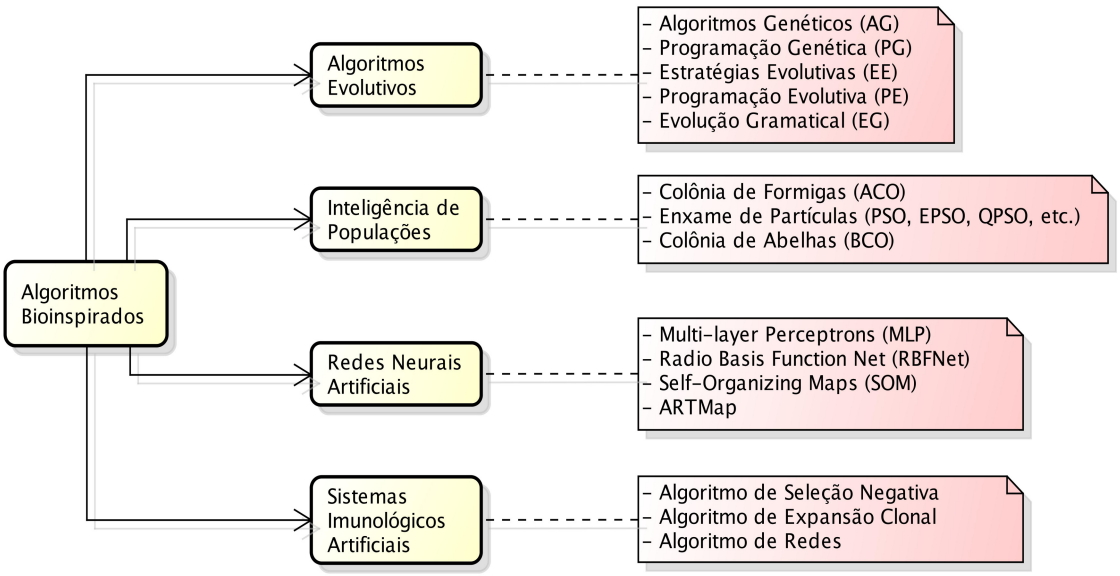
A computação bioinspirada é um sub-ramo da computação natural, que utiliza características encontradas na natureza com o objetivo de extrair conceitos, princípios e mecanismos fundamentados em processos biológicos para serem aplicados em sistemas computacionais [Teixeira et al. 2011, Dorigo et al. 1996, Goldberg 1989, Medeiros et al. 2005]. Esta área de pesquisa multidisciplinar investiga como computadores podem ser utilizados para modelar características de sistemas biológicos e como as soluções encontradas por ela podem originar novos paradigmas de computação.

Diversos problemas envolvendo busca e otimização podem ser encontrados no mundo real e podem ser definidos através de regras com as quais se deseja maximizar ou minimizar valores de uma determinada função numérica (restritiva ou não) ou manipular problemas de tempo exponencial, tais quais os do tipo NP-completos (caxeiro viajante, soma dos subconjuntos) [Filho et al. 2008]. Estes problemas podem ser enquadrados em uma gama de aplicações prática na indústria, no gerenciamento de suplementos básicos como energia elétrica, gás e refinamento de petróleo [Linden 2008, Teixeira et al. 2011].

A computação tem buscado inspirações na natureza para lidar com estes tipos de problemas, sendo que diversos cientistas da computação como Von Newman, Norbert Wiener e Alan Turing foram motivados à construção de sistemas de computação com capacidade adaptativa ao ambiente presente [Teixeira 2012, Mitchell 1999]. Esses cientistas buscavam inspiração na biologia e em outras áreas de conhecimento como a psicologia e a eletrônica para a construção de soluções mais eficientes [Yasojima et al. 2009].

O conhecimento obtido em diversas áreas da ciência a respeito de sistemas naturais (i.e. exatas, biológicas) já vem sendo utilizadas com considerável sucesso para o desenvolvimento de ferramentas tecnológicas capazes de resolver problemas de grande complexidade, cujas soluções eram até então, desconhecidas ou computacionalmente impraticáveis [Dorigo & Gambardella 1997, Medeiros et al. 2005]. De acordo com [Teixeira 2012, Filho et al. 2008, Forbes 2005, de Castro 2010], o uso de algoritmos inspirados em sistemas biológicos ou físico tem por base o estudo e desenvolvimento de metaheurísticas[1] capazes de solucionar problemas ou obter soluções satisfatórias em relação a técnicas clássicas de otimização (e.g. programação linear). A Figura 1.1 mostra as diversas categorias e tipos de algoritmos bioinspirados.

Figura 1.1: Classificação dos Algoritmos Bioinspirados



[1] Segundo [Teixeira 2012], uma metaheurística consiste em um conjunto de conceitos e métodos usados para definir procedimentos heurísticos, gerando assim uma estrutura genérica de um algoritmo que pode ser aplicada em diferentes problemas de busca e otimização com o objetivo de encontrar uma solução praticável - não necessariamente ótima - em tempo de processamento viável.

Como exemplos de sistemas computacionais baseados em sistemas naturais têm-se a computação evolucionária [Back et al. 2000], os sistemas imunológicos artificiais [Dasgupta 1999, de Castro & Timmis 2002], as redes neurais artificiais [Haykin 1999], inteligência de enxames [Bonabeau et al. 1999], o algoritmo de arrefecimento simulado [Aarts & Korst 1989] e mais recentemente, os algoritmos culturais [Reynolds & Peng 2005].

Dentre as metaheurísticas bioinspiradas mais utilizadas, as baseadas no modelo de população têm apresentado desempenho satisfatório em solucionar problemas de otimização onde se apresenta um cenário de uma ou de múltiplas soluções [Eberhart & Shi 2000]. Um algoritmo que vem sendo bastante adotado é o de otimização por enxame de partículas (PSO), que é inspirado no comportamento de uma revoada de pássaros para encontrar uma solução ótima em um espaço limitado de busca [Kennedy & Eberhart 1995]. Na literatura, o PSO é classificado com um algoritmo que utiliza a abordagem de *Swarm Intelligence*, ou Inteligência de Enxames [Haupt & Haupt 1998, Kennedy & Eberhart 2001].

O algoritmo PSO é conhecido por ser uma metaheurística com várias modificações documentadas, Dentre elas, destacam-se duas dessas variações utilizadas neste trabalho:

- **EPSO - Otimização por Enxame de Partículas Evolucionário**: Proposto por [Leite et al. 2010, Miranda & Fonseca 2002], essa variante traz o conjunto de operações de estratégias evolucionárias como replicação, mutação e seleção aplicado em um ambiente PSO modificado;
- **QPSO - Otimização por Enxame de Partículas com Inspiração Quântica**: Proposto por [Sun et al. 2004*b*], o modelo de inspiração quântica traz várias modificações ao algoritmo PSO original, onde as partículas se movimentam seguindo regras baseadas na mecânica quântica ao invés da clássica com movimentação inspirada na mecânica Newtoniana.

O PSO também apresenta diversas implementações que envolve o conceito de multipopualção ou multi-enxames, tendo como destaque os trabalhos concebidos por [den Bergh & Engelbrecht 2004, Niu et al. 2007] para o desenvolvimento da abordagem utilizada nesta dissertação. Essa classe de algoritmos consiste na utilização de dois ou mais enxames sendo executados de maneira competitiva ou cooperativa, cuja interação e estrutura variam de acordo com a topologia utilizada. Em geral, grande parte dessas metaheurísticas [e.g. Niu et al. 2005] utilizam sistemas computacionas com arquitetura distribuída e paralela, cuja função é a divisão da carga de processamento em um ambiente inter-comunicável entre dois ou mais enxames. Como resultado, ambientes multi-enxames auxiliam na obtenção de resultados mais eficientes, dada a troca de informações entre si [Niu et al. 2007, El-Abd & Kamel 2008].

Com base em diversos trabalhos publicados [e.g. Veronese & Krohling 2009, Souza et al. 2011, Mussi et al. 2011, Souza et al. 2013, Souza et al. 2014], pode-se afirmar que a implementação do PSO e de suas variantes com recursos de estratégia evolucionária em um modelo competitivo e multienxame, aplicado sob o paradigma GPGPU traz uma série de benefícios, considerando que cada partícula pode ser tratada por uma *thread*, que ao ser aplicada em um ambiente de paralelismo massivo de múltiplos enxames independentes, apresenta um considerável aumento de desempenho e performance em relação ao tempo de execução, bem como no processo de busca e otimização.

Neste trabalho, são propostos dois novos algoritmos, baseados nas metaheurísticas EPSO e QPSO sob uma abordagem multi-enxame, inspirados em um modelo de topologia mestre-escravos encontrado na literatura [e.g. Niu et al. 2007, El-Abd & Kamel 2008] e aplicados sob a arquitetura CUDA, denominados de *Competitive Evolutionary Multi-Swarm Optimization on Graphics Processing Units* (CEMSO) e *Competitive Quantum Evolutionary Multi-Swarm Optimization on Graphics Processing Units* (CQEMSO).

1.2 GPUs Como Unidade de Processamento Geral

A partir da concepção das primeiras GPUs no início dos anos 80 com o lançamento iSBX 275 *Video Graphics Controller Multimodule Board* da Intel, uma série de mudanças significativas ocorreram em seu desenvolvimento. Foram descobertas novas funcionalidades, e principalmente, um grande aumento na sua capacidade de processamento que culminou na primeira década do século XXI em sua utilização para outros propósitos que não fossem gráficos, surgindo assim o paradigma GPGPU aplicado em ambientes de computação heterogênea [2] [Souza 2010].

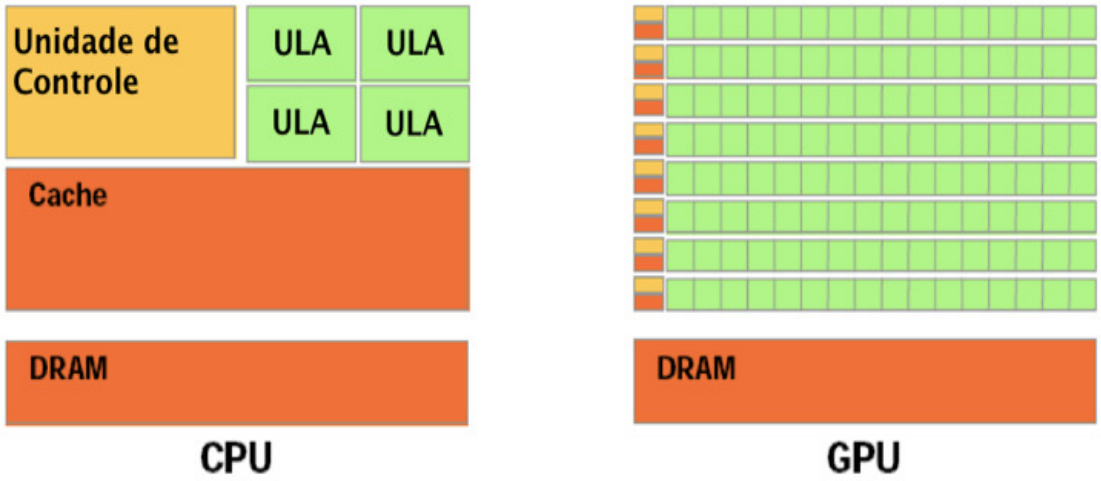
- **Unidades de Processamento Multi-Núcleo *(Multi-Core)***: É o modelo adotado principalmente pelas principais empresas fabricantes de CPUs, em especial aquelas que estão inseridas no mercado de processadores da arquitetura x86. Essa vertente busca o aumento de velocidade dos programas sequenciais através da divisão das instruções entre os núcleos. Nesse cenário, uma execução de um programa *multithread* seria feito através da distribuição dos dados do programa entre essas *thread*s e executadas em núcleos distintos. Cada núcleo possui seu próprio *cache* podendo assim, executar várias instruções simultaneamente;

[2] A computação heterogênea é classificada como um conjunto de sistemas computacionais que possuem diferentes tipos de unidades de processamento, onde geralmente se aplicam processadores especializados no intuito de acelerar a execução de determinada operação, além de permitir o tipo de processador mais adequado para cada aplicação específica [Stone et al. 2010, Costa 2013]

- **Unidades de Processamento Com Milhares de Núcleos *(Many-Core)***: Empresas como NVIDIA e AMD adotam a abordagem de processadores com milhares de núcleos de processamento (ou *many-core*) em suas GPUs. Essa vertente possui o foco apontado para a execução de aplicações paralelas de forma massiva, cujo o objetivo é distribuir e processar de forma totalmente simultânea os dados disponíveis que não possuem dependência funcional ou de dados [Kirk & Hwu 2010].

A Figura 1.2 mostra de forma simplificada o esquema físico de uma CPU (abordagem *Single-Core* e *Multi-Core*) e uma GPU (abordagem *Many-Core*) destacando a distribuição de componentes como as Unidades Lógica e Aritmética (ULA), Unidades de Controle (UC) e outras partes fundamentais que compõem uma unidade de processamento.

Figura 1.2: Estrutura física simplificada de uma CPU e uma GPU. Adaptado de:[Kirk & Hwu 2010, NVIDIA 2012*a*, NVIDIA 2008*a*]

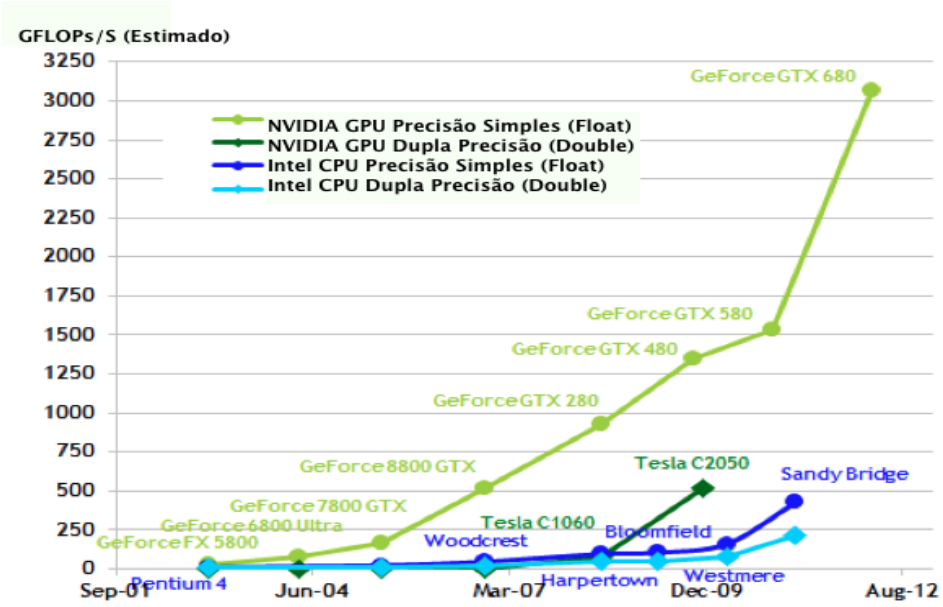


A abordagem geral nos primeiros anos de uso da GPU para processamento geral era muito restrita, dado que a única maneira de interagir com a GPU era através de programação utilizando as APIs (*Application Programming Interface*) gráficas (OpenGL e DirectX)[Sanders & Kandrot 2010], sendo que a execução de cálculos ou programas de propósito geral em uma GPU estariam sujeitos às restrições de programação dessas APIs. Devido a tais limitações, os pesquisadores começaram a explorar computação de propósito geral executando aplicações sobre as APIs gráficas como se fosse uma operação tradicional de processamento de imagem [Sanders & Kandrot 2010, Kirk & Hwu 2010].

Mesmo com essas limitações, os processadores *many-core*, fortemente representado pelas GPUs, tem mostrado superioridade em termos de velocidade e aproveitamento de paralelismo, principalmente com operações envolvendo dados de ponto flutuante simples. Tal desempenho compensava a utilização, mesmo sem uma ambiente de programação adequado até 2007, quando a NVIDIA lançou oficialmente a arquitetura CUDA, que contribuiu para uma adoção ainda maior do paradigma GPGPU, bem como sua consolidação, por vários centros de pesquisa e universidades pelo mundo [Hwu et al. 2008, NVIDIA 2012*a*].

Para demonstrar a capacidade de processamento das GPUs em relação as CPUs, a Figura 1.3 mostra a comparação das placas de vídeo NVIDIA e AMD em relação aos processadores convencionais da arquitetura x86.

Figura 1.3: Comparação de desempenho entre CPU e GPU para operações com ponto flutuante por segundo. Fontes:[Kirk & Hwu 2010, NVIDIA 2012*a*]



Com o surgimento de ambientes voltados para a computação de propósito geral em GPUs, diversas soluções computacionais, com destaque para aplicações da área científica [e.g. Mathworks 2013, CST 2013] vem sendo desenvolvidas, tirando proveito de processamento dos milhares de núcleos disponíveis nos multiprocessadores encontrados em uma placa de vídeo através de tecnologias como CUDA, OpenACC e OpenCL [Kirk & Hwu 2010].

1.3 Motivação

O processo de otimização pode ser descrito como um procedimento de detecção de atributos e configuração de parâmetros de um sistema, cujo o objetivo é produzir melhores resultados [Parsopoulos & Vrahatis 2010, Haupt & Haupt 1998, Eberhart & Simpson 1996]. Eis alguns exemplos de como a otimização baseada em metaheurísticas bioinspiradas pode ajudar na execução de tarefas em diversas áreas do conhecimento:

- **Engenharia Estrutural**: Detecção do melhor projeto possível para a produção de uma estrutura segura e econômica, que esteja dentro das normas de segurança;
- **Ciência da Computação**: Modelagem e projeto de ambientes computacionais de alto desempenho ao menor custo;
- **Modelagem Operacional**: Identificação da melhor configuração possível para as linhas de produção no intuito de maximizar a eficiência.

Na maioria dos problemas, sistemas complexos são modelados com funções multi-dimensionais restritivas que não podem ser facilmente resolvidos por otimização linear ou qualquer solução combinatória, sendo que em certos casos, o tempo de processamento é economicamente e funcionalmente inviável (e.g. Problema do Caixeiro Viajante [Linden 2008]). De maneira geral, a escolha do algoritmo PSO e de suas variantes se deve ao fato de que os mesmos possuem características desejáveis para o processo de otimização desenvolvido nesta dissertação, tais como:

- São metaheurísticas baseadas em um modelo de população, o que torna possível a realização uma exploração por diversas soluções ao mesmo tempo, além de possuir mecanismos voltados para o tratamento de restrições;
- O PSO tem ganhado amplo reconhecimento e utilização devido sua eficiência na obtenção de soluções, aliado a uma implementação robusta e simples [Parsopoulos & Vrahatis 2010], com aplicações em diversos problemas como treinamento de redes neurais artificiais [Kuok et al. 2010], controle de sistemas de lógica Fuzzy [Hsuan-Ming 2005], dentre outros;
- Ao ser comparado com algoritmos genéticos, o PSO destaca-se pela a facilidade de implementação e quantidade reduzida de parâmetros a ser configurados;
- Estabelecem a manutenção de diversidade, mantendo cada proposta de solução em uma região diferente do espaço de busca;
- São estruturados de maneira a permitir um alto grau de paralelismo, desde o nível de indivíduo até o nível de enxame;

Além dos pontos citados acima, pode-se afirmar com base em trabalhos encontrados na literatura [e.g. Veronese & Krohling 2009, Petr et al. 2010, Mussi et al. 2011, Solomon et al. 2011] que a implementação de algoritmos de otimização baseado em sistemas bioinspirados atrelado ao uso de unidades de processamento paralelo, como a GPU, vem se mostrando uma área de pesquisa com resultados promissores.

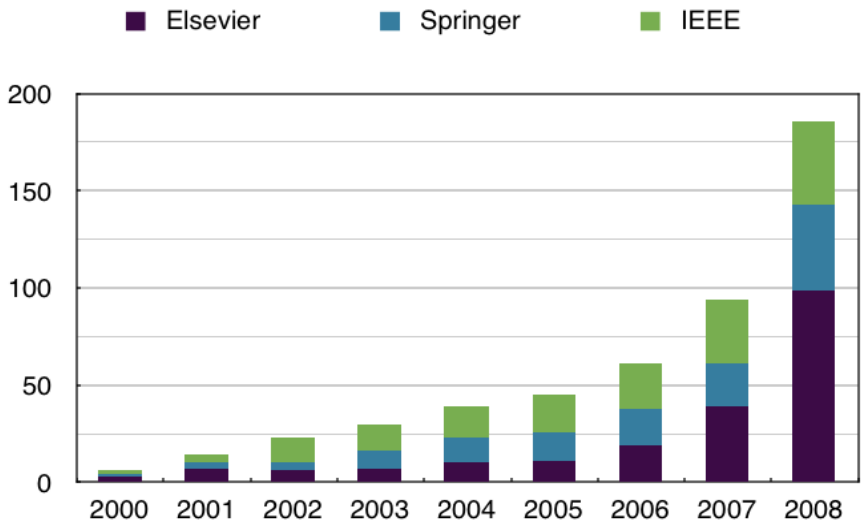
A principal motivação deste trabalho consiste na utilização das estratégias evolutivas encontradas no algoritmo EPSO aplicadas aos algoritmos QPSO e PSO clássico, além de estender suas funções para um modelo multi-enxame e cooperativo, isso permitiria a aplicação de mecanismos como migração de partículas, troca de informações com base em modelos competitivos ou cooperativos, dentre outros procedimentos.

1.3.1 Popularidade dos Algoritmos PSO e Variantes

Nos últimos doze anos, o PSO vem ganhando amplo reconhecimento e utilização devido a sua eficiência na obtenção de soluções, aliado a uma implementação robusta e simples [Parsopoulos & Vrahatis 2010]. Dado o potencial do PSO para paralelização explícita, bem como a sua capacidade de adaptação dos componentes e operadores a assumir uma forma desejada implícita ao problema em questão, o PSO e suas variantes tem se apresentado como uma alternativa popular entre algoritmos de otimização.

No intuito de demonstrar o aumento de popularidade do PSO, a Figura 1.4 ilustra a quantidade de artigos em publicações científicas envolvendo o PSO em três grandes instituições (Elsevier, Springer e IEEE) entre os anos de 2000 e 2008.

Figura 1.4: Aumento da quantidade de trabalhos envolvendo o algoritmo PSO em três grandes editoras. Adaptado de:[Parsopoulos & Vrahatis 2010]



A quantidade de trabalhos mostrada na Figura 1.4 não se deve apenas a utilização do algoritmo PSO, mas também as variações e modificações, que tem como objetivo aumentar a performance, contribuindo na busca de melhores resultados obtidos para uma determinada gama de problemas. Dentre as variações, destacam-se o EPSO e o QPSO, onde ambos tem demonstrado bons resultados em relação ao PSO clássico para uma gama de problemas de otimização [e.g. Leite et al. 2010, Sun et al. 2004*b*, Xi et al. 2007].

Dada as informações expostas, a principal motivação deste trabalho consiste na utilização das estratégias evolutivas encontradas no algoritmo EPSO aplicadas aos algoritmos QPSO e PSO clássico, além de estender suas funções para um modelo multi-enxame e cooperativo, o que nos permitiria a aplicação de mecanismos como migração de partículas, troca de informações com base em modelos competitivos ou cooperativos, dentre outros procedimentos.

1.4 Objetivos e Principais Contribuições

O principal objetivo deste trabalho é o estudo, modelagem e implementação de dois novos algoritmo, baseado nos mecanismo evolutivos encontrados no EPSO aplicados no PSO e QPSO, denominados *Particle Swarm Optimization With Evolutionary Strategies (PSO+EE)* e *Quantum-Behaviour Particle Swarm Optimization With Evolutionary Strategies (QPSO+EE)*. As soluções geradas são utilizadas como base para a implementação de uma abordagem multi-enxame e competitiva sob a arquitetura paralela CUDA, gerando assim mais dois algoritmos de otimização: *Competitive Evolutionary Multi-Swarm Optimization (CEMSO)* e *Competitive Quantum-Behaviour Evolutionary Multi-Swarm Optimization (CQEMSO)*. Outros pontos fundamentais no desenvolvimento deste trabalho estão na execução de ambientes multi-enxames em computadores pessoais, em tempo hábil através da arquitetura CUDA; aplicação de estratégias evolutivas nos algoritmos QPSO e PSO sob uma topologia de multi-enxame mestre-escravos e a concepção de novas condições de contorno para refinamento da busca (equivalência e espelhamento).

As contribuições advindas do desenvolvimento deste trabalho são a hibridização dos modelos clássico e quântico do algoritmo PSO com a abordagem de estratégias evolutivas encontradas no EPSO, concepção e documentação de dois algoritmos híbridos bioinspirados (i.e. CEMSO e CQEMSO), análise de parâmetros e configuração de maior eficência para busca e otimização de problemas restritivos voltados a engenharia, estrutura de implementação de algoritmos bioinspirados sob o paradigma GPGPU.

1.4.1 Objetivos Específicos

Esse trabalho possui os seguintes objetivos específicos:

- Produção de material para o aprimoramento dos algoritmos implementados, bem como a utilização do material desenvolvido em outras metaheurísticas.
- Compreender os principais conceitos referentes ao funcionamento dos algoritmos EPSO e QPSO, suas definições e implementação;
- Produção de documentação envolvendo as principais metaheurísticas baseadas em otimização por enxame de partículas aplicadas em uma abordagem multi-enxame de topologia mestre-escravos;
- Oferecer um modelo baseado na arquitetura CUDA para a implementação de outros algoritmos bioinspirados que visam uma execução concorrente e modelos multi-populacional;
- Contribuir na produção e referência de material em língua portuguesa que possa ser utilizado como ponto de partida para estudo e pesquisas futuras sobre GPGPU e CUDA aplicada em soluções baseadas em computação natural, com ênfase em algoritmos bioinspirados baseados em inteligência de enxames;

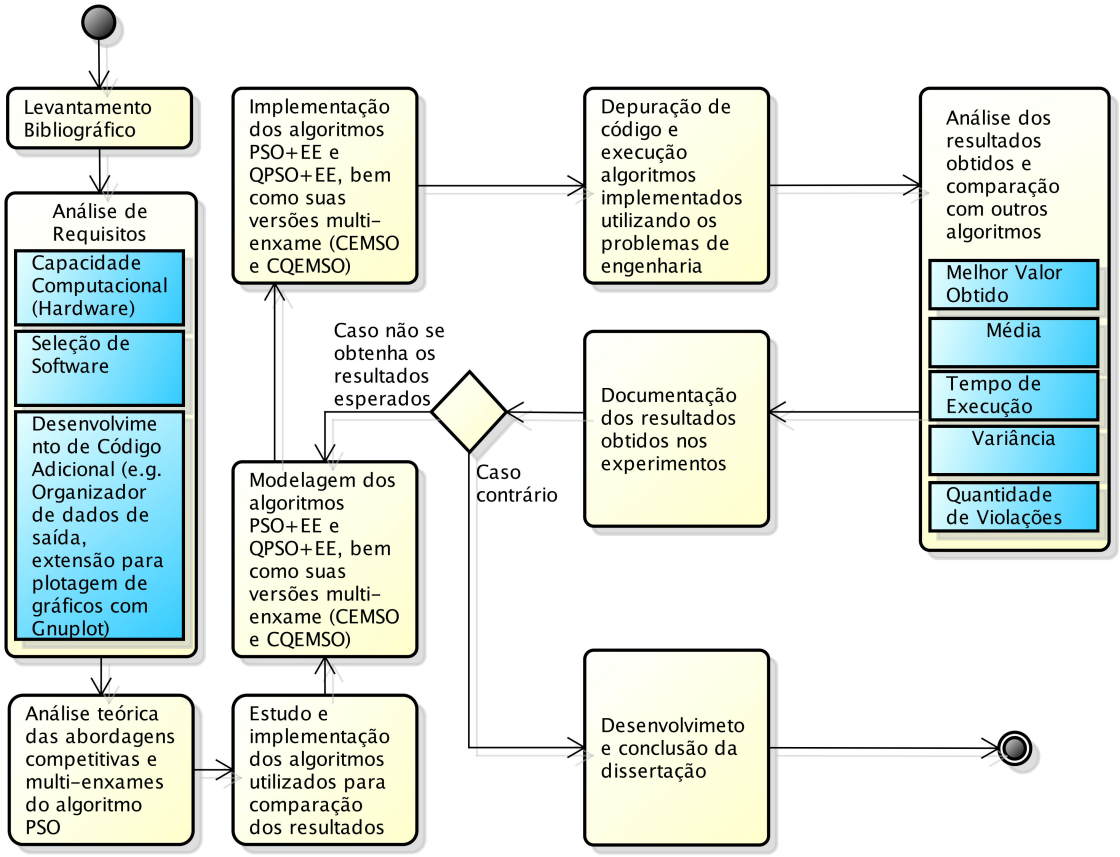
1.5 Metodologia

Para a elaboração deste trabalho, foi realizada uma ampla pesquisa bibliográfica acerca dos temas abordados no intuito de adquirir um embasamento maior nas áreas estudadas, além de buscar pontos convergentes entre os assuntos (computação bioinspirada (com ênfase em PSO, QPSO e EPSO) e computação paralela através do paradigma GPGPU). O passo seguinte consiste na análise de requisitos e seleção das máquinas a serem utilizadas nos testes de desempenho. O uso da arquitetura CUDA deve-se a diversos fatores, onde se destacam o baixo custo de *hardware*, alto grau de paralelismo envolvendo milhares de *threads*, facilidade de programação e a utilização de computadores pessoais, sem a necessidade de *clusters* ou supercomputadores.

Durante o desenvolvimento deste trabalho, foram produzidos pseudo-códigos no intuito de facilitar a visualização de cada etapa de execução do algoritmo e por fim, a implementação do algoritmoss propostos (CEMSO e CQEMSO). Além dos algoritmos CEMSO, CQEMSO e de suas respectivas extensões mono-enxame (PSO+EE e QPSO+EE), foram implementados sob a arquitetura CUDA os algoritmos PSO, EPSO, QPSO e suas respectivas versões multi-enxame com topologia mestre-escravos (COMSO, COEMSO e COQMSO).

Para um melhor entendimento do processo, a Figura 1.5 mostra os estágios de desenvolvimento deste trabalho.

Figura 1.5: Fluxograma de atividade da metodologia utilizada para o desenvolvimento deste trabalho



1.5.1 Problemas de Engenharia e Procedimentos de Validação

Quando um problema sujeito a restrições é manipulado, sua representação é matemáticamente realizada por um conjunto de funções que satisfazem uma série de condições. De acordo com [Silva et al. 2012], tais restrições são importantes ao representar problemas de projeto de engenharia, uma vez que normalmente são impostas na descrição inicial do problema e que às vezes são difíceis de satisfazer, tornando assim a busca ineficiente. Nestes casos, faz-se necessário a utilização de métodos com processo de busca refinado e customizado, onde obtem-se uma quantidade maior de resultados que satisfaçam todas as restrições.

Para a avaliação do algoritmo, quatro funções de otimização para problemas de engenharia (restritirvas) foram usadas para os testes de desempenho. Segue abaixo as funções relacionadas aos problemas de engenharia:

- Projeto de Viga de Aço (WBD);
- Minimização do Peso da Tensão/Compressão Sobre Mola (MWTCS);
- Projeto de Vaso de Pressão (DPV);
- Projeto de Redutor de Velocidade de 11 Restrições (SRD11);

Os algoritmos desenvolvidos foram submetidos a um processo de análise de desempenho, bem como uma constante revisão, tanto do modelo quanto da implementação no intuito de corrigir erros e aprimorar recursos. Os dados coletados nesta fase são avaliados através da média, variância, tempo de execução e do melhor resultado obtido durante uma quantidade pré-determinada de experimentos. É imporante ressaltar que os resultados obtidos foram comparados com resultados de outros trabalhos publicados e que utilizam as mesmas funções de engenharia.

1.5.2 Estrutura da Dissertação

Este trabalho é composto por seis capítulos. Além do capítulo introdutório, esta dissertação esta dividida nos seguintes tópicos:

- ***Capítulo 2***: A fundameção teórica apresenta os principais aspectos teóricos dos algoritmos bioinspirados utilizados neste trabalho (e.g. PSO, EPSO, QPSO), bem como do ambiente de programação paralela em GPUs CUDA.
- ***Capítulo 3***: Apresenta os algoritmos CEMSO e CQEMSO. São mostrados os algoritmos propostos, agrupando cada trabalho e sua contribuição com as soluções desenvolvidas; os trabalhos relacionados; as características técnicas, algoritmos PSO e QPSO com técnicas de estratégias evolutivas baseadas no algoritmo EPSO (PSO+EE, QPSO+EE); as estruturas multi-enxame com destaque para a topologia baseada na comunicação mestre-escravos aplicada em um ambiente com estratégias evolutivas e a organização dos dados e instruções sob a arquitetura CUDA;
- ***Capítulo 4***: São expostos os testes realizados utilizando por base quatro problemas de engenharia (restritivos), além dos comentários e análise dos resultados obtidos;
- ***Capítulo 5***: Apresenta as considerações finais sobre o trabalho e propostas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta o estado da arte nos assuntos relacionados a computação paralela atrelada ao uso de GPUs para processamento de dados geral sob a arquitetura CUDA, a plataforma computacional paralela utilizada nos algoritmos CEMSO e CQEMSO. Encontra-se também neste capítulo, as principais características dos algoritmos PSO, EPSO e QPSO, além da estrutura multi-enxame com topologia mestre-escravos e operações não encontradas nos algoritmos originais (e.g. condições de contorno por esquema *"Damping"*, fator de decrescimento α).

2.1 Programação de Propósito Geral Para GPUs (CUDA)

A arquitetura CUDA (*Compute Unified Device Architecture*) foi concebida pela NVIDIA em meados de 2006 e oficialmente lançada em 2007, através das placas de vídeo GeForce da série 8. Consiste em um ambiente de programação onde um conjunto de instruções permitem que aplicações paralelas sejam executadas em GPUs, permitindo um ambiente de programação heterogênea entre duas unidades de processamento (CPU e GPU) [Stringhini et al. 2012, Kirk & Hwu 2010].

Diferentemente das placas de gerações anteriores, onde os recursos de computação eram divididos e representados na forma de variáveis contendo dados voltados para processamento gráfico, as placas de vídeo compatíveis com a arquitetura CUDA utilizam *shaders* [1] unificados, permitindo que cada unidade lógica e aritmética (ULA) presente na placa possa ser utilizada por aplicações voltadas para cálculos e operações de propósito geral [Sanders & Kandrot 2010].

[1] Os *shaders* são instruções de software usadas nas GPUs durante o processo de renderização. São divididas em três unidades: Objetos (*Geometry Shader*), vértices (*Vertex Shader*) e pixels (*Pixel Shader*).

As ULAs das placas NVIDIA são compatíveis com o padrão IEEE 754 [2] [NVIDIA 2012*a*, IEEE 2007], porém, com restrições envolvendo operações de ponto flutuante de dupla-precisão. Tirando proveito das principais características do modelo de processamento de fluxo, a arquitetura CUDA cria um ambiente de programação geral amigável com interoperabilidade entre APIs gráficas, mas que requer do programador, um conhecimento apurado sobre os recursos, além de outras funcionalidades que tiram vantagem de um ambiente de paralelismo massivo. Até o fechamento dessa dissertação, a arquitetura CUDA se encontrava na versão 6.5.

O desenvolvimento de aplicativos em CUDA segue o modelo de paralelismo SIMT [e.g. NVIDIA 2008*b*, Stringhini et al. 2012], o que permite grande desempenho em problemas que envolvam uma quantidade elevada de dados independentes, como matrizes de alta dimensionalidade, operações com matrizes esparsas, inversão de matrizes, etc.

2.1.1 Organização de *Threads*

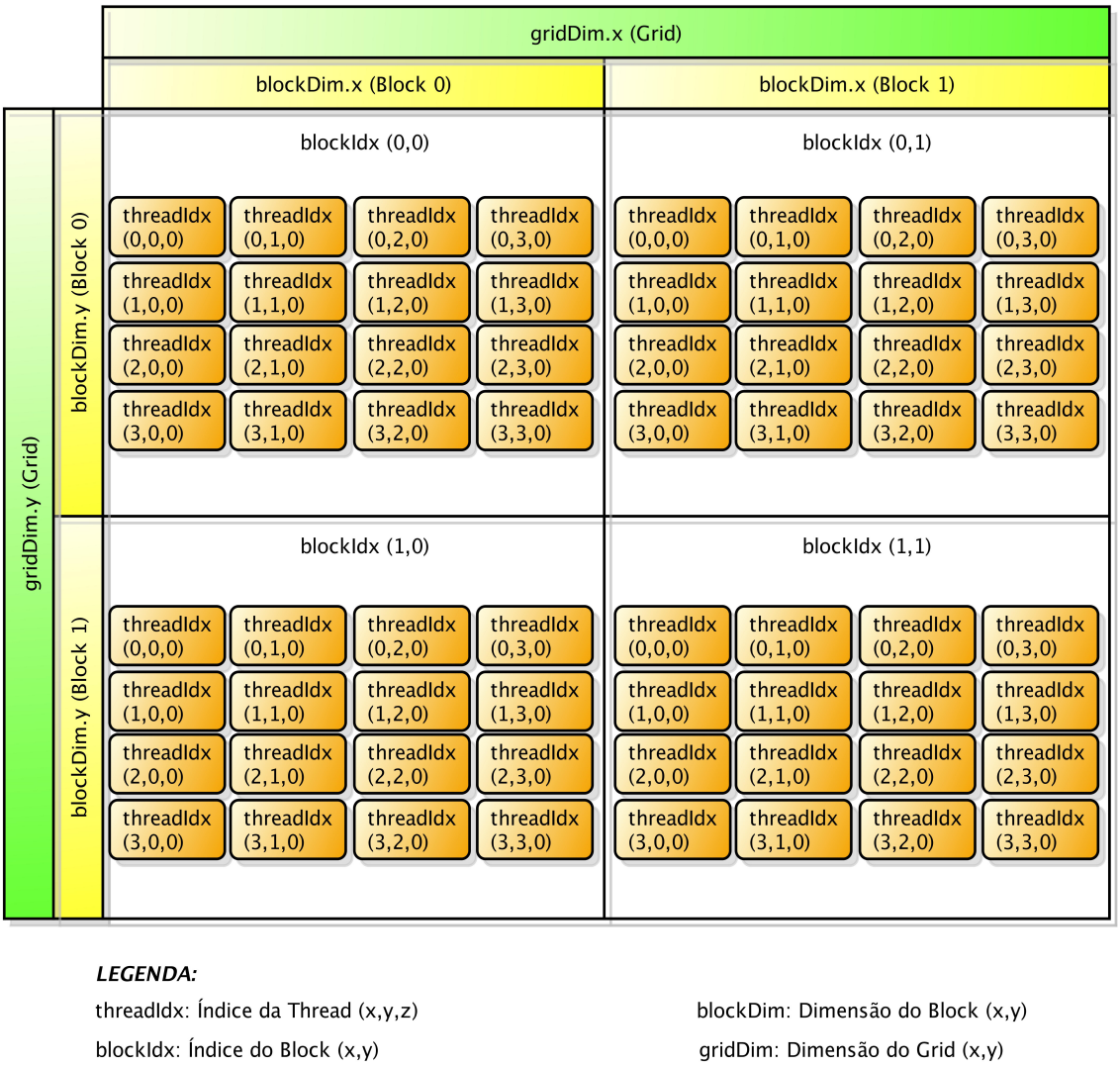
Para manter a organização do fluxo de execução paralela aliada a uma distribuição de dados e controle de acesso a memória eficaz, a arquitetura CUDA divide o contexto de execução em três hierarquias de *threads*:

- ***Thread***: São as unidades básicas de execução da arquitetura CUDA. Cada *thread* fica responsável por uma cópia do programa para uma determinada quantidade de dados e os executa paralelamente, desde um único elemento de um vetor ou matriz, até operações que envolvam quantidades de dados maiores não dependentes. Em algumas GPUs da série Kepler, como a GeForce GT 650M, é possível obter até 2048 *threads* por multiprocessador de fluxo SMX [NVIDIA 2012*b*];
- ***Block***: Essa estrutura armazena um vetor ou uma matriz (2D ou 3D) de *threads*, sendo que todos os *blocks* possuem quantidades iguais de *threads*. Através dessa organização, é possível utilizar sincronismo de *threads* a nível de *blocks*, ou seja, todas as *threads* de um *blocks* podem ser sincronizadas através de um comando específico [NVIDIA 2012*b*]. As GPUs anteriores a série Fermi abrigavam até 512 *threads* por *block*, e as superiores suportam até 1024 *threads* por *block*;
- ***Grid***: Conjunto de todos os *blocks* utilizado por uma função executada na GPU. Um *grid* pode ser organizado como um vetor ou matriz (apenas 2D) de *blocks*.

[2] Padronização de operações envolvendo números com ponto flutuante de precisão simples e dupla (float e double respectivamente) [IEEE 2007]

A Figura 2.1 mostra o exemplo de uma estrutura de hierarquia bidimensional de *threads*, *blocks* e *grids* em uma GPU.

Figura 2.1: Hierarquia bidimensional de *threads* em CUDA. Adaptado de:[NVIDIA 2012*a*]



A organização das *threads* no contexto da arquitetura CUDA atua na divisão do fluxo de execução em duas ou mais tarefas, sendo que o programador possui a capacidade de configurar o número de *threads*, *blocks* e *grids* necessários para a execução de um programa. Por padrão, a menos que sejam determinadas em instruções condicionais para sincronização, a ordem de execução das *threads* é determinada pelo hardware de forma assíncrona.

2.1.2 Estrutura Básica de um Programa em CUDA-C

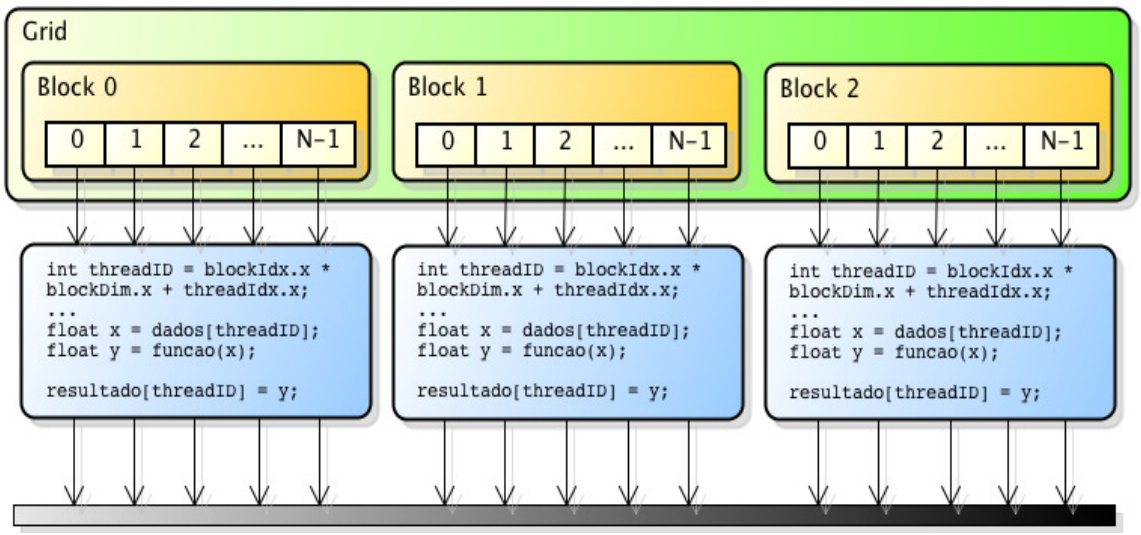
CUDA possui dois tipos distintos de código, um para execução na CPU (*host*) e outro na GPU (*device*). O código *host*, escrito em ANSI C, agrega a parte sequencial do programa, transferência de dados entre CPU e GPU e operações cujo o melhor tempo de execução ocorre na CPU. Já o código *device* contém a parte paralela do programa, sendo escrito em uma versão extendida do padrão ANSI C, com instruções que são usadas para criar as funções executadas na GPU [Kirk & Hwu 2010, Souza 2010].

As funções utilizadas na GPU para processamento geral são classificadas como:

- **Funções *Kernel* (`__global__`)**: São as principais funções em CUDA-C, pois agregam todo o código paralelo executado na GPU. Cada ”cópia” dessa função é distribuída entre as *threads*, onde serão executadas utilizando dados distintos;
- **Extensões de *Kernel* (`__device__`)**: Essas funções são criadas especialmente para uso exclusivo dos *kernels*, pois o interpretador reconhece qualquer função sem identificador como uma função pertencente ao código *host* [Souza 2010].

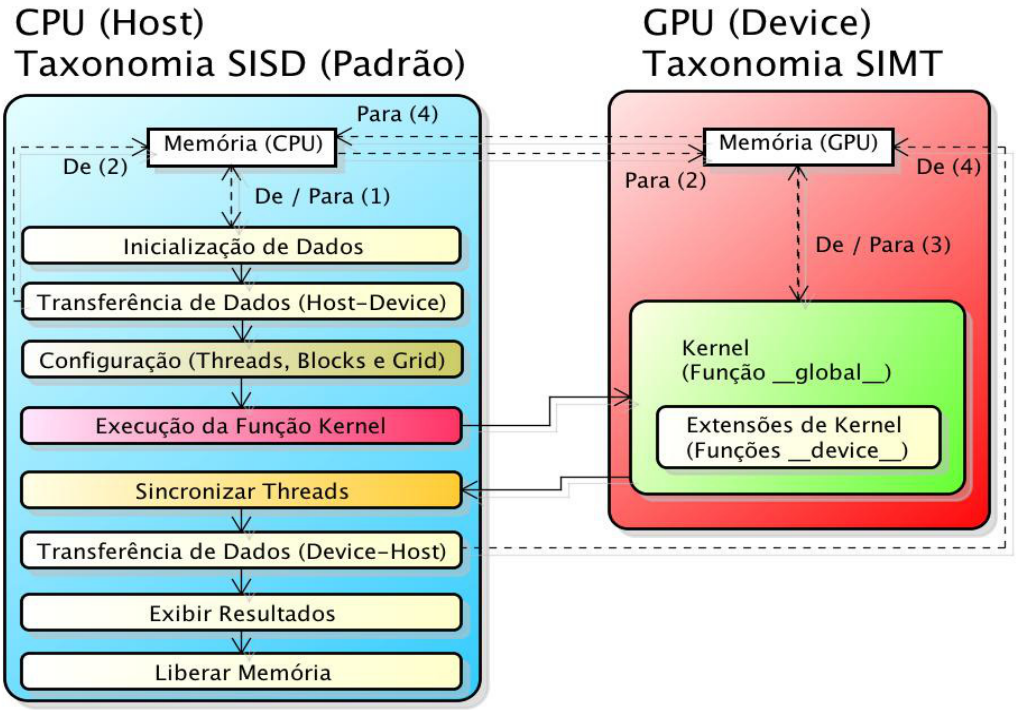
Para exemplificar o funcionamento de um programa CUDA, a Figura 2.2 mostra a distribuiço de *threads* em relação ao *kernel*.

Figura 2.2: Esquema de distribuição de *threads* no *kernel*



Através das funções de *Kernel* e suas extensões, é possível criar um programa em CUDA-C com pouco esforço de programação, o que é uma das vantagens dessa arquitetura em relação a outros métodos de computação paralela como MPI [Sanders & Kandrot 2010]. Para melhor compreender o fluxo de processamento de uma aplicação escrita sob a arquitetura CUDA, a Figura 2.3 mostra a estrutura básica de um programa, desde as operações de transferência de dados entre CPU e GPU, até a chamada de *kernel* pelo código *host*.

Figura 2.3: Estrutura básica de um software implementado sob a arquitetura CUDA



Dentre as principais vantagens da arquitetura CUDA destacam-se, a capacidade de implementação de algoritmos com alto nível de pararelismo em poucas linhas de código, ambiente de programação gratuito com hardware básico de baixo custo em relação a outros ambientes de processamento paralelo e distribuído, boa documentação, compatibilidade com outras soluções voltada para computação heterogênea (e.g. OpenCL, OpenACC, DirectCompute) [Stringhini et al. 2012].

A partir do capítulo 3, será abordado o uso da arquitetura CUDA aplicada nos algoritmos propostos (CEMSO e CQEMSO), aproveitando os principais recursos que esta pode proporcionar para a implementação de algoritmos bioinspirados.

2.2 *Particle Swarm Optimization*

A metaheurística do enxame de partículas PSO (*Particle Swarm Optimization*) é uma técnica de otimização estocástica, inicialmente voltada para funções contínuas, desenvolvida e aprimorada por James Kennedy e Russel Eberhart [Kennedy & Eberhart 1995]. Foi concebida a partir de estudos relacionados ao ”comportamento social” observado em algumas espécies de pássaros e outros animais (i.e. cardume de peixes), tendo como base os trabalhos de observação comportamental do biólogo Frank Heppner [Pomeroy 2003]. É pertencente ao grupo dos algoritmos bioinspirados e inseridos na categoria de algoritmos baseados em inteligência de enxames ou populações [Engelbrecht 2007].

O PSO apresenta similaridades com técnicas de computação evolucionária, aplicadas em busca e otimização baseada no processo de seleção natural descrito por Charles Darwin [Darwin 1859]. De acordo com [Filho et al. 2008], a avaliação de qualidade da população em geral pode ser realizada por uma função de avaliação (*fitness*) que irá atribuir notas de performance, resultando posteriormente em um processo de seleção dos melhores indivíduos [Teixeira 2012, Souza 2010]. Através de mecanismos que provoquem alterações nesta população, objetivando assim a formação de novas soluções que irão contribuir para a evolução da população.

A principal diferença entre o PSO e as técnicas de computação evolucionária é o modelo de busca aplicado na população. Enquanto nos algoritmos evolucionários, uma nova geração de indivíduos surge, a princípio, através de processos evolutivos (i.e. mutação, seleção natural), o PSO trabalha com um modelo de busca em um espaço n-dimensional através de cálculos para novos valores de velocidade (inclui variáveis aleatórias na equação) e posição baseados em elementos da mecânica clássica.

2.2.1 Implementação

A idéia básica do algoritmo PSO clássico está no uso de conjunto de indivíduos que interagem através da troca inteligente de informações sobre os melhores valores obtidos por cada partícula (local, representados pela variável P_i) e o melhor obtido pelo enxame como um todo (global, representado pela variável P_g) buscando num espaço delimitado, resultados de melhor qualidade, podendo ou não, atingir o ótimo global. Baseado em um modelo de exploração coletiva e cooperativa, as partículas tendem a progredir no processo de busca a medida que novos e melhores resultados são encontrados.

A partícula é uma estrutura que contém variáveis relacionadas ao seu posicionamento, velocidade e valor atribuído por uma função de avaliação (*fitness*), sendo o enxame, um conjunto de partículas agregadas em um espaço de busca. Cada partícula possui valores de posição que representam a solução do problema, sendo armazenados em uma linha ou coluna de matriz, possibilitando o acesso e a ornização dos dados de cada partícula.

Interação Individual e Social

As constantes C_1 e C_2, são aplicadas na equação de atualização de velocidade, cuja principal função é a configuração da relevância de interação cognitiva ou social do enxame, em outras palavras, representam respectivamente o fator de interação individual e fator de correção social aplicado no enxame [Eberhart & Shi 2000]. O comportamento do enxame é diretamente relacionado a proporção entre os valores de interação individual e social. Valores altos de C_1 em relação a C_2, indicam que o algoritmo dará mais importância ao conhecimento individual da partícula do que ao conhecimento global.

Fator de Inércia

Incorporado posteriormente no algoritmo PSO original por [Eberhart & Shi 1998, Eberhart & Shi 2001], o fator de inércia (w) tem como principal objetivo a melhoria na taxa de convergência do resultado, ou seja, a capacidade com que as partículas do enxame encontram a solução ótima, ou algum resultado muito próximo dessa solução.

No PSO, o valor do fator de inércia tende a decrescer conforme o comportamento selecionado pelo usuário, sendo o decrescimento linear e de fatores estocástico os mais utilizados [Eberhart & Shi 1998, Eberhart & Shi 2000, Shi & Eberhart 1999, Filho et al. 2008, Niu et al. 2007]. As Equações 2.1 e 2.2 demonstram o comportamento do fator de inércia com decrescimento linear e estocástico, respectivamente.

$$w = \frac{(k-1)}{(I_{MAX}-1)(-W_{MAX}+W_{MIN})+W_{MAX}} \tag{2.1}$$

$$w = 0.5 - \frac{rand}{2} \tag{2.2}$$

As variáveis descritas nas Equações 2.1 e 2.2 são: Fator de inércia (w); Iteração atual (k); Quantidade máxima de iterações (I_{MAX}); Limite mínimo e máximo para o fator de inércia (W_{MIN} *e* W_{MAX}); Número aleatórios gerado uniformemente entre 0 até 1 (*rand*).

Atualização de Velocidade e Posição

As partículas circulam pelo espaço de busca, tendo suas velocidades atualizadas de forma dinâmica com base no histórico das experiências individuais e coletiva de todo o enxame. Logo, a evolução do algoritmo do PSO está associada à trajetória percorrida pelo enxame e ao tempo gasto para encontrar a melhor solução do problema. As estruturas de armazenamento dos valores de velocidade (V) e posição (X) de cada partícula do enxame em uma matriz são descritas pelas Equações 2.3 e 2.4.

$$V_{[i][j]} = \left[V_{[i][0]}, V_{[i][1]}, V_{[i][2]}, V_{[i][3]}, ..., V_{[i][n-1]}\right]^{N_{PAR}} \tag{2.3}$$

$$X_{[i][j]} = \left[X_{[i][0]}, X_{[i][1]}, X_{[i][2]}, X_{[i][3]}, ..., X_{[i][n-1]}\right]^{N_{PAR}} \tag{2.4}$$

As variáveis descritas nas Equações 2.3 e 2.4 são: Quantidade de Partículas (N_{PAR}); Índice da partícula (i); Índice da variável (j).

Com as estruturas de posição e velocidade configuradas, inicia-se o processo de otimização descritos nas Equações 2.5 (velocidade) e 2.6 (posição).

$$V_i(t+1) = wV_i(t) + R_1C_1(P_i - X_i(t)) + R_2C_2(P_g - X_i(t)) \tag{2.5}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{2.6}$$

As variáveis descritas nas Equações 2.5 e 2.6 são: Tempo atual da partícula (t); Fator de inércia (w); Índice da partícula (i); Valor atual de velocidade ($V_i(t)$); Novo valor de velocidade ($V_i(t+1)$); Valor atual de posição ($X_i(t)$); Novo valor de posição ($X_i(t+1)$); Melhor solução encontrada pela partícula (mínimo/máximo local) (P_i); Melhor solução encontrada pelo enxame (mínimo/máximo global) (P_g); Fatores de interação individual e social (C_1 e C_2); Número aleatórios gerados uniformemente entre de 0 até 1 (R_1 *e* R_2).

No PSO, os valores de posição ($X_i(t)$) e velocidade ($V_i(t)$) devem estar inseridos dentro dos limites mínimo (X_{MIN} e V_{MIN}) e máximo (X_{MAX} e V_{MAX}) de cada variável presente na partácula. Esses limites determinam o espaço de busca do enxame, evitando que as partículas gerem resultados inválidos. Tais limites são descritos pelas Equações 2.7 e 2.8.

$$X_{MIN} \leq X_i(t) \leq X_{MAX} \tag{2.7}$$

$$V_{MIN} \leq V_i(t) \leq V_{MAX} \tag{2.8}$$

Condições de Contorno de Posição e Velocidade (”*Damping*”)

Quando uma partícula escapa do espaço de busca em uma das dimensões, ela é reposicionada no limite do espaço de busca violado e seu componente de velocidade naquela dimensão é redirecionado para a direção oposta, tendo seu valor multiplicado por um número aleatório gerado no intervalo real de 0 e 1. É importante ressaltar que parte da velocidade é perdida devido ao reflexo imperfeito, sendo que o valor da perda é determinado aleatoriamente. A Equação 2.9 descreve o processo de correção de velocidade.

$$V_i(t+1) = \begin{cases} V_{MIN} & \text{se } V_i(t+1) < V_{MIN} \\ V_i(t+1) & \text{se } V_{MIN} \leq V_i(t+1) \leq V_{MAX} \\ V_{MAX} & \text{se } V_i(t+1) > V_{MAX} \end{cases} \tag{2.9}$$

As Equações 2.10 e 2.11 descrevem o processo de correção de posição baseado na técnica ”*Damping*”.

$$V_i(t+1) = \begin{cases} -V_i(t+1)R_1 & \text{se } X_i(t+1) < X_{MIN} \\ V_i(t+1) & \text{se } X_{MIN} \leq X_i(t+1) \leq X_{MAX} \\ -V_i(t+1)R_1 & \text{se } X_i(t+1) > X_{MAX} \end{cases} \tag{2.10}$$

$$X_i(t+1) = \begin{cases} X_{MIN} & \text{se } X_i(t+1) < X_{MIN} \\ X_i(t+1) & \text{se } X_{MIN} \leq X_i(t+1) \leq X_{MAX} \\ X_{MAX} & \text{se } X_i(t+1) > X_{MAX} \end{cases} \tag{2.11}$$

A Figura 2.4 ilustra o processo de reposicionamento ”*Damping*”.

Figura 2.4: Ilustração do procedimento de contorno ”*Damping*”. Adaptado de:[Xi et al. 2007]

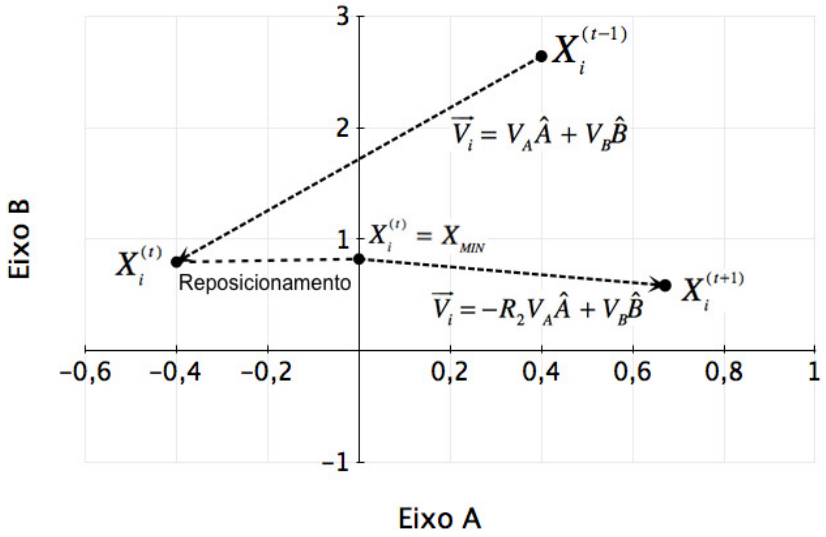
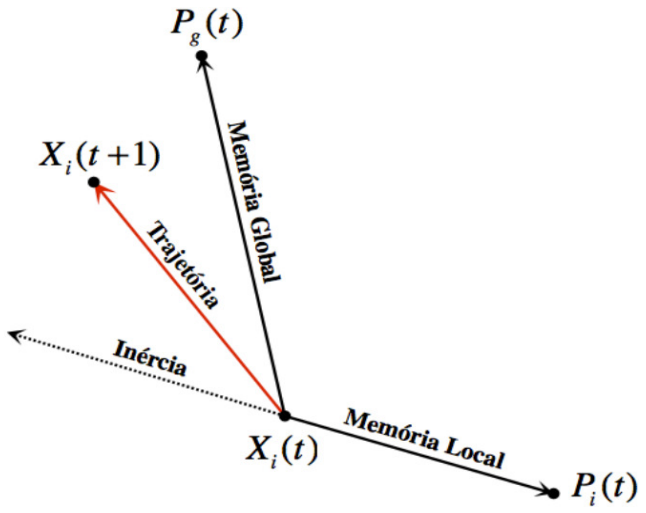


Ilustração de Movimento e Pseudo-Código

A Figura 2.5 ilustra as forças envolvidas na movimentação da partícula, destacando os vetores de memória (exerce atração sobre a partícula para o melhor local (P_i)), cooperação (exerce atração sobre a partícula para o melhor global (P_g)) e inércia.

Figura 2.5: Ilustração de movimentação da partícula no PSO



O Algoritmo 1 demonstra os procedimentos do PSO com fator de inércia.

Algoritmo 1: PSO clássico com fator de inércia

```
Alocar memória para estruturas do enxame (posição (X), velocidade (V));
Alocar memória para melhores valores do enxame (local (P_i), global (P_g));
for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
    for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
        Inicializar os valores de velocidade;
        Inicializar os valores de posição;
    Avaliar partícula (Fitness);
    Inicializar os melhor local da partícula (P_i);
    Obter o melhor global do enxame (P_g);
for iter ← 1 to (QUANTIDADE DE ITERAÇÕES−1 do
    Atualizar fator de inércia (Equação 2.1 ou 2.2);
    for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
        for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
            Atualizar valor de velocidade da variável j (Equação 2.5);
            Aplicar ajuste de velocidade (Equação 2.9);
            Atualizar valor de posição da variável j (Equação 2.6);
            Aplicar condições de contorno "Damping" aos valores de posição (Equação 2.11);
            Aplicar condições de contorno "Damping" aos valores de velocidade (Equação 2.10);
        Avaliar partícula (Fitness);
        Obter os melhor local da partícula (P_i);
        Obter o melhor global do enxame (P_g);
```

2.3 *Evolutionary Particle Swarm Optimization*

O algoritmo EPSO foi desenvolvido por [Miranda & Fonseca 2002] e consiste em uma metaheurística que une elementos de estratégias evolutivas (EE) em um PSO modificado, em que os operadores de recombinação clássica são substituídos pelas regras do movimento das partículas. Segundo [Leite et al. 2010, Naing 2008], do ponto de vista conceitual, o EPSO permite uma dupla interpretação sobre sua funcionalidade, podendo ser visto sob duas perspectivas:

- Algoritmo de inteligência de enxame híbrido. Uma variante do algoritmo PSO;
- Algoritmo evolutivo híbrido, classificado como uma variante das EE.

Ao contrário do PSO, onde os valores de interação individual e social são constantes ajustadas pelo usuário e o fator de inércia tem seu valor definido por uma equação de decrescimento ou estocástica (vide Equação 2.1 e 2.2), o algoritmo EPSO utiliza valores que são modificados no decorrer do processo de otimização, sendo inicializados através de um gerador de números radômicos com intervalo entre zero e um, submetendo-os posteriormente a um processo de mutação a cada iteração. Tal operação auxilia no processo de busca do algoritmo, pois a utilização de valores dinâmicos aos fatores social e individual tende a impedir que as partículas fiquem ”presas” em melhores locais[3].

Réplicas

As réplicas no algoritmo EPSO são geradas após o processo de movimentação das partículas originais. O principal objetivo desta estratégia é uma busca mais eficiente e com maior variabilidade. A estrutura de armazenamento de dados das réplicas é similar a das partículas originais em termos de posição e velocidade. Tais estruturas para os valores de velocidade (rV) e posição (rX) de cada réplica são descritas pelas Equações 2.12 e 2.13:

$$rV_{[i][j]} = \left[rV_{[i][0]}, rV_{[i][1]}, rV_{[i][2]}, rV_{[i][3]}, ..., rV_{[i][n-1]}\right]^{N_{PAR}} \tag{2.12}$$

$$rX_{[i][j]} = \left[rX_{[i][0]}, rX_{[i][1]}, rX_{[i][2]}, rX_{[i][3]}, ..., rX_{[i][n-1]}\right]^{N_{PAR}} \tag{2.13}$$

[3]Em [Miranda & Fonseca 2002], os autores afirmam que o PSO pode apresentar perda no desempenho de busca caso o valor do fator de inércia apresente baixa variação entre as iterações ou atinja o valor mínimo pré-estabelecido pelo usuário.

Mutação (Fator de Inércia, Interação Social e Individual)

Durante a execução do algoritmo, as réplicas geradas em cada iteração tem seus valores de inércia, interação social e individual mutados. Ao contrário do algoritmo PSO, cada partícula possui um valor próprio de fator de inércia (w_i), bem como de interação individual ($C_{1(i)}$) e social ($C_{2(i)}$). As Equações 2.14 a 2.16 demonstram a estrutura de w_i, $C_{1(i)}$ e $C_{2(i)}$ de cada partícula original do enxame aplicado em um vetor.

$$w_{[i]} = [w_{[0]}, w_{[1]}, w_{[2]}, w_{[3]}, ..., w_{[n]}] \tag{2.14}$$

$$C_{1[i]} = [C_{1[0]}, C_{1[1]}, C_{1[2]}, C_{1[3]}, ..., C_{1[n]}] \tag{2.15}$$

$$C_{2[i]} = [C_{2[0]}, C_{2[1]}, C_{2[2]}, C_{2[3]}, ..., C_{2[n]}] \tag{2.16}$$

As Equações 2.17, 2.18 e 2.19 demonstram o processo de mutação do fator de inércia, interação individual e social respectivamente.

$$mw_{(i)} = w_i + (1 + \sigma N(0,1)) \tag{2.17}$$

$$mC_{1(i)} = C_{1(i)} + (1 + \sigma N(0,1)) \tag{2.18}$$

$$mC_{2(i)} = C_{2(i)} + (1 + \sigma N(0,1)) \tag{2.19}$$

As variáveis descritas nas Equações 2.14, 2.15, 2.16, 2.17, 2.18 e 2.19 são: Índice da partícula (i); Índice da variável (j); Fator de inércia submetido ao processo de mutação para a réplica da partícula i ($mw_{(i)}$); Interação individual submetido ao processo de mutação para a réplica da partícula i ($mC_{1(i)}$); Interação social submetido ao processo de mutação para a réplica da partícula i ($mC_{2(i)}$); Valor atual do fator de inércia da partícula i (w_i); Valor atual do fator de interação individual da partícula i ($C_{1(i)}$); Valor atual do fator de interação social da partícula i ($C_{2(i)}$); Valor de pertubação para os fatores de inércia, interação individual e social (σ); Variável aleatória com distribuição gaussiana com valores de desvio padrão e variância iguais a 0 e 1, respectivamente ($N(0,1)$).

Pertubação do Melhor Global

Outra característica importante do algoritmo EPSO é o processo de pertubação do melhor global original do enxame (P_g) e a geração de um melhor global alterado ($mP_{g(i)}$). Esta operação auxilia o algoritmo de modo a evitar que as partículas fiquem "presas" em melhores locais. O objetivo deste processo é, assim como a mutação das variáveis w_i, $C_{1(i)}$ e $C_{2(i)}$, proporcionar uma variabilidade maior no processo de busca no enxame ao oferecer uma nova referência de melhor global para as réplicas. De acordo com [Miranda & Fonseca 2002], pode-se descrever o processo de pertubação da seguinte forma:

- Se o ótimo global já tiver sido encontrado, o processo de otimização não será afetado e o algoritmo segue seu fluxo de execução até o final;
- Senão, os processos de mutação e pertubação do melhor global permitem que a busca seja focalizada em uma área específica e não apenas tendo como referência o melhor global original.

As Equações 2.20 e 2.21 demonstram o processo de de geração do fator de pertubação ω_i e o procedimento de pertubação do melhor global original.

$$\omega_i = \omega_i + (1 + \sigma_g N(0,1)) \tag{2.20}$$

$$mP_{g(i)} = P_g + (1 + \omega_i N(0,1)) \tag{2.21}$$

As variáveis descritas nas Equações 2.20 e 2.21 são: Melhor global submetido ao processo de pertubação da partícula i ($mP_{g(i)}$); Melhor global original (P_g); Constante de pertubação para o melhor global (σ_g); Fator de pertubação da partícula i (ω_i); Variável aleatória com distribuição gaussiana com valores de desvio padrão e variância iguais a 0 e 1, respectivamente ($N(0,1)$).

Atualização de Velocidade e Posição

A atualização de velocidade do algoritmo EPSO aplicada as réplicas tem por base uma versão modificada da equação utilizada pelo algoritmo PSO, onde a variação do fator de inércia, e as constantes de interação individual e social são substituídos pelos valores submetidos ao processo de mutação de $C_{1(i)}$, $C_{2(i)}$ e pertubação do melhor global. Para as partículas originais, as constantes de interação individual e social são substituídas por variaveis com valores diferentes para cada partícula. As Equações 2.22 e 2.23 mostram o cálculo de atualização de velocidade de uma partícula original e réplica respectivamente.

$$V_i(t+1) = \begin{cases} w_{(i)}V_i(t) + C_{1(i)}(P_i - X_i(t)) + C_{2(i)}(mP_{g(i)} - X_i(t)) & \text{se } rand < \theta \\ w_{(i)}V_i(t) + C_{1(i)}(P_i - X_i(t)) & \text{se } rand > \theta \end{cases} \tag{2.22}$$

$$rV_i(t+1) = \begin{cases} mw_{(i)}V_i(t) + mC_{1(i)}(P_i - X_i(t)) + mC_{2(i)}(mP_{g(i)} - X_i(t)) & \text{se } rand < \theta \\ mw_{(i)}V_i(t) + mC_{1(i)}(P_i - X_i(t)) & \text{se } rand > \theta \end{cases} \tag{2.23}$$

As novas posições são calculadas através das Equações 2.24 (partícula original) e 2.25 (réplica). É importante lembrar que os valores de posição da partícula (original ou réplica) estão delimitados pelos limites mínimo e máximo (X_{MIN} e X_{MAX}).

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{2.24}$$

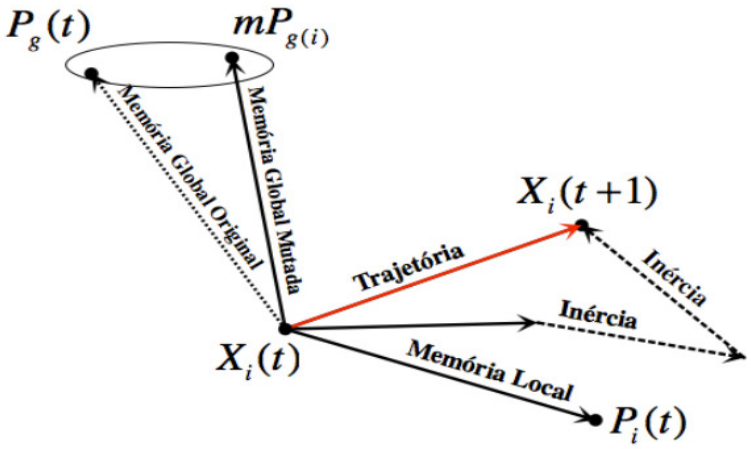
$$rX_i(t+1) = X_i(t) + rV_i(t+1) \tag{2.25}$$

As variáveis descritas nas Equações 2.20 e 2.21 são: Fator de inércia de partícula i (w_i); Fator de inércia mutada e aplicada a réplica da partícula i ($mw_{(i)}$); Constante de probabilidade aplicada a topologia de estrela estocástica [Miranda et al. 2007] θ; Valor atual de velocidade para partícula i ($V_i(t)$); Valor atual de velocidade atribuída a réplica da partícula i ($rV_i(t)$); Novo valor de velocidade para partícula i ($V_i(t+1)$); Valor atual de posição da partícula i ($X_i(t)$); Valor atual de posição atribuída a réplica da partícula i ($rX_i(t)$); Melhor solução encontrada pela partícula (mínimo/máximo local) (P_i); Melhor solução encontrada pelo enxame (mínimo/máximo global) (P_g); Melhor global submetido ao processo de mutação ($mP_{g(i)}$); Fatores de correção individual e social da partícula i ($C_{1(i)}$ *e* $C_{2(i)}$); Fatores de interação individual e social mutados (réplica da partícula i) ($mC_{1(i)}$ *e* $mC_{2(i)}$).

Ilustração de Movimento e Pseudo-Código

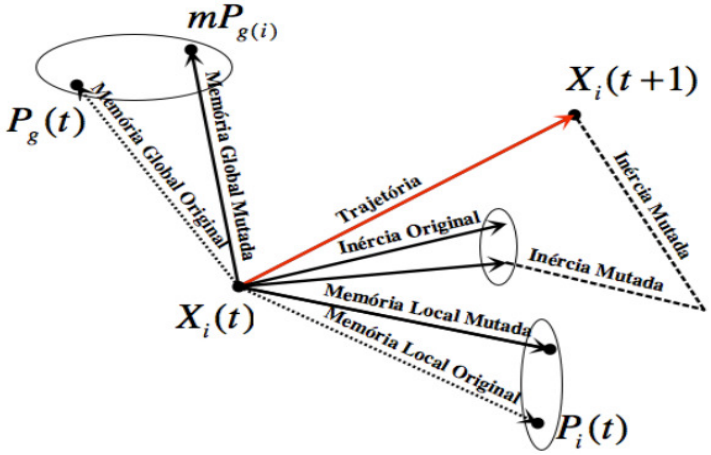
A Figura 2.6 ilustra as forças envolvidas na movimentação da partícula original no algoritmo EPSO, destacando os vetores de memória (exerce atração sobre a partícula para o melhor local (P_i)), cooperação (exerce atração sobre a partícula para o melhor global (P_g)) e inércia.

Figura 2.6: Ilustração de movimentação da partícula no EPSO (partículas originais)



A Figura 2.7 ilustra as forças envolvidas na movimentação das réplicas no algoritmo EPSO, destacando os vetores de memória (exerce atração sobre a partícula para o melhor local (P_i)), cooperação (exerce atração sobre a partícula para o melhor global (P_g)) e inércia. Os vetores aplicados as réplicas são os mutados, sendo os originais apenas valores de referência para o processo de mutação, descritos pelas Equações 2.17 a 2.19.

Figura 2.7: Ilustração de movimentação da partícula no EPSO (réplicas)



O Algoritmo 2 demonstra os procedimentos do EPSO com condições de contorno.

Algoritmo 2: EPSO com condições de contorno

```
Alocar memória para estruturas do enxame (posição (X), velocidade (V));
Alocar memória para melhores valores do enxame (local (P_i), global (P_g));
Alocar memória para fator de interação individual (C_1(i));
Alocar memória para fator de interação social (C_2(i));
Alocar memória para fator de inércia (w_(i));
for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
    for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
        Inicializar os valores de velocidade;
        Inicializar os valores de posição;
    Inicializar os valores de fator de inércia;
    Inicializar os valores de interação individual;
    Inicializar os valores de interação social;
    Inicializar os valores para o fator de pertubação do melhor global;
    Avaliar partícula (Fitness);
    Inicializar os melhor local da partícula (P_i);
    Obter o melhor global do enxame (P_g);
for iter ← 1 to (QUANTIDADE DE ITERAÇÕES−1 do
    Atualizar fator de inércia (Equação 2.1 ou 2.2);
    for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
        for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
            Atualizar valor de velocidade da variável j para a partícula original (Equação 2.22);
            Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
            Atualizar valor de posição da variável j para a partícula original (Equação 2.6 ou 2.24);
            Aplicar condições de contorno "Damping" aos valores de posição para a partícula original
            (Equação 2.11);
            Aplicar condições de contorno "Damping" aos valores de velocidade para a partícula
            original (Equação 2.10);
        Avaliar partícula original (Fitness);
        for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
            Gerar novo valor para o fator de inércia atravé da mutação (2.17));
            Gerar novo valor para o fator de interação individual através da mutação (2.18));
            Gerar novo valor para o fator de interação social através da mutação (2.19));
            Gerar novo valor para o fator de pertubação atravé da mutação (2.20));
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a réplica k (Equação 2.23);
                Aplicar ajuste de velocidade para a réplica k (Equação 2.9);
                Atualizar valor de posição da variável j para a réplica k (Equação 2.25);
                Aplicar condições de contorno "Damping" aos valores de posição para a réplica k
                (Equação 2.11);
                Aplicar condições de contorno "Damping" aos valores de velocidade para a réplica k
                (Equação 2.10);
            Avaliar réplica (Fitness);
            Comparar réplica e partícula original;
            if fitness(Replica) melhor do que fitness(Original) then
                Substituir partícula original pela réplica (Original = Replica);
                Substituir fatores C_1(i), C_2(i) e w_(i) por mC_1(i), mC_2(i) e mw_(i);
                Substituir fitness original por fitness réplica;
        Obter os melhor local da partícula (P_i);
        Obter o melhor global do enxame (P_g);
```

2.4 *Quantum-Behaviour Particle Swarm Optimization* (QPSO)

O PSO de inspiração quântica (QPSO) proposto por [Sun et al. 2004*b*], é um algoritmo de otimização originalmente concebido para ser um sistema de busca mais complexo do que o enxame de partículas com inspiração Newtoniana (PSO). De acordo com [Xi et al. 2007], uma representação linear de otimização como a encontrada no PSO original não é suficiente para representar a evolução de um enxame, dado que o processo de otimização baseado em um sistema de enxames envolve eventos de natureza indeterminística [4], sendo melhor representado através de um esquema baseado em sistemas quânticos.

Com o desenvolvimento da mecânica quântica iniciado nos primeiros anos do século XX com base nos trabalhos desenvolvidos por físicos como Max Planck, Louis de Broglie, Erwin Schrödinger, Werner Heisenberg, Niels Bohr e David Bohn, os cientistas foram forçados a repensar a aplicabilidade da mecânica clássica e da compreensão tradicional da natureza de movimentos de objetos microscópicos [Pang 2005, Vignati et al. 2004, Xi et al. 2008].

No PSO clássico, o movimento da partícula é descrito pelo seus valores de posição (X_i) e vetor de velocidade (V_i) que determinam sua trajetória seguindo a mecânica Newtoniana. Contudo, considerando um sistema baseado em mecânica quântica, uma trajetória descrita de forma vetorial passa a ser irrelevante, dado que os valores X_i e V_i de uma partícula não podem ser determinados simultaneamente de acordo com o princípio de incerteza de Heisenberg [Herrera 2007]. Com base em [Sun et al. 2004*a*], pode-se enumerar algumas vantagens do algoritmo QPSO em relação ao PSO clássico.

- Os sistemas quânticos atuam sob o Princípio da Superposição Quântica [5], de modo que o mesmo possui mais estados possíveis do que um sistema Newtoniano. Pode-se ilustrar tal sistema pelo experimento mental do "Gato de Schrödinger" [6];
- Sistemas quânticos são baseados em incerteza, diferentemente do sistema estocástico clássico. Antes da medição, uma partícula pode estar em qualquer posição com uma distribuição de probabilidade, consequência de ter a trajetória indeterminada.

[4] De acordo com a Interpretação de Copenhagen clássica, o indeterminismo em sistemas quânticos consiste na incapacidade de saber ou prever de forma "fixa" os comportamentos das partículas pela alteração do observador, bem como de seus estados anteriores a medição [Vignati et al. 2004].

[5] Princípio da mecânica quântica onde um sistema existe parcialmente em todos os estados possíveis antes de ser medido. Após a medição, o sistema passa a ter um único estado [Vignati et al. 2004].

[6] Experimento mental proposto por Erwin Schrödinger para demonstrar as possibilidades na superposição quântica: Só saberemos se o gato está vivo ou morto apoós abrir a caixa, mas se isso for feito, será alterada a possibilidade resultado final [Schrodinger 1935].

2.4.1 Implementação

No modelo quântico do PSO, o estado de uma partícula é representado através da função de onda $\Psi(X_i)$, ao contrário de uma descrição determinística de posição e velocidade. A probabilidade da partícula aparecer na posição X_i é calculada através da função de densidade $|\Psi(X_i)|^2$, a qual depende diretamente do campo potencial em que a partícula se encontra [Xi et al. 2007, Sun et al. 2004*a*]. As Equações 2.26 e 2.27 mostram a função de onda da partícula ($\Psi(X_i)$) e a densidade de probabilidade ($|\Psi(X_i)|^2$) respectivamente.

$$\Psi(X_i(t)) = \frac{1}{\sqrt{L_i(t)}} e^{\frac{-|p_i - X_i(t)|}{L_i(t)}} \tag{2.26}$$

$$|\Psi(X_i(t))|^2 = \frac{1}{L_i(t)} e^{\frac{-2|p_i - X_i(t)|}{L_i(t)}} \tag{2.27}$$

As variáveis descritas nas Equações 2.26 e 2.27 são: Índice da partícula (i); Tempo atual da partícula i (t); Valor atual de posição da partícula i ($X_i(t)$); Função de onda da partícula i ($\Psi(X_i)$); Densidade de probabilidade da partícula i ($|\Psi(X_i)|^2$); Fator de criatividade; [7] (L); Valor de referência de inclinação para a partícula X_i (p).

Learning Inclination Point (*LIP*)

Durante o processo de otimização, existem dois tipos de informações que influem no comportamento de uma partícula no QPSO: Melhor local (P_i) e melhor global (P_g). O ponto p_i encontrado na função de onda é conhecido como LIP (*Learning Inclination Point*) do individuo. Trata-se de um valor localizado entre P_i e P_g, onde a tendência de busca de cada indivíduo faz com que cada um procure na vizinhança de seu *LIP* [Sun et al. 2004*a*, Sun et al. 2004*b*]. A Equação 2.28 demonstra o cálculo do *LIP*.

$$LIP_i = \frac{(R_{(c1)} P_i) + (R_{(c2)} P_g)}{(R_{(c1)} + R_{(c2)})} \tag{2.28}$$

As variáveis descritas nas Equações 2.26 e 2.27 são: Índice da partícula (i); Melhor solução encontrada pela partícula (mínimo/máximo local) (P_i); Melhor solução encontrada pelo enxame (mínimo/máximo global) (P_g); Números aleatórios gerados uniformemente no intervalo de 0 até 1 ($R_{(c1)}$ *e* $R_{(c2)}$).

[7] Segundo [Herrera 2007, Mikki & Kishk 2008], o valor de L depende da quantidade de energia do campo potencial, o qual define o escopo de procura da partícula

Média de Melhores Locais e Fator α

No intuito de se obter resultados mais equilibrados para o fator de criatividade (L) em relação aos melhores locais (P_i), [Sun et al. 2004a] introduziu o cálculo de média dos melhores locais (P_{Me}). Em [Xi et al. 2008], o coeficiente de ponderação de média local (fator α) se apresenta como uma variável de decrescimento linear ou estocástica, sendo inserida no cálculo de média dos melhores locais. Se for aplicado um comportamento decrescente ao fator α, maior será seu valor, caso a partícula esteja perto da melhor solução [Xi et al. 2008]. As Equações 2.29, 2.30 e 2.31 mostram o fator α com decrescimento linear e estocástico, bem como a média dos melhores locais com o fator α (P_{Me}), respectivamente.

$$\alpha = \frac{(k-1)}{(I_{MAX}-1)(-\alpha_{MAX}+\alpha_{MIN})+\alpha_{MAX}} \tag{2.29}$$

$$\alpha = 0.5 - \frac{rand}{2} \tag{2.30}$$

$$P_{Me} = \frac{1}{N}\sum_{i=0}^{N-1}\alpha_i P_i = \left(\frac{1}{N}\sum_{i=0}^{N-1}\alpha_{io}P_{i0}, \frac{1}{N}\sum_{i=0}^{N-1}\alpha_{i1}P_{i1}, ..., \frac{1}{N}\sum_{i=0}^{N-1}\alpha_{id}P_{id}\right) \tag{2.31}$$

As variáveis descritas nas Equações 2.26 e 2.27 são: Índice da partícula e de iteração (i *e* k); Quantidade de partículas no enxame (N); Quantidade máxima de iterações (I_{MAX}); Limites mínimo e máximo para o fator α (α_{MIN} *e* α_{MAX}); Melhor solução encontrada pela partícula (mínimo/máximo local) (P_i); Fator de ponderação de melhores locais (α_i).

Fator de Contração-Expansão (β)

Durante a atualização dos valores de posição, é necessário manter o balanceamento entre a qualidade da busca e a taxa de convergéncia do algoritmo. No QPSO, esse controle é feito através do fator de contração-expansão (β). As Equações 2.32 e 2.33 mostram o cálculo do fator de contração-expansão (β) linear e estocástico.

$$\beta = \frac{(k-1)}{(I_{MAX}-1)(-\beta_{MAX}+\beta_{MIN})+\beta_{MAX}} \tag{2.32}$$

$$\beta = 0.5 - \frac{rand}{2} \tag{2.33}$$

As variáveis descritas nas Equações 2.32 e 2.33 são: Limite mínimo e máximo para o fator de contração-expansão (β_{MIN} e β_{MAX}); Quantidade máxima de iterações (I_{MAX}); índice de iteração (k); Fator de contração-expansão (β).

Parâmetro de Criatividade (L)

Denomina-se parâmetro de criatividade ou de imaginação (L), o escopo básico de busca da partícula, onde seu valor é diretamente proporcional a probabilidade da partícula descobrir novos valores de posição [Sun et al. 2004*a*, Xi et al. 2008]. A Equação 2.34 mostra o cálculo do fator de criatividade (L).

$$L_i(t+1) = 2\beta \left| P_{Me} - X_i(t) \right| \tag{2.34}$$

As variáveis descritas na Equação 2.34 são: Novo parâmetro de criatividade da partícula i ($L_i(t+1)$); Fator de contração-expansão (β); Valor atual de posição da partícula i ($X_i(t)$); Soma da média dos melhores locais (P_{Me}).

Equação de Atualização de Posição

Através dos cálculos de média dos melhores locais, parâmetro de criatividade e *Learning Inclination Point*, é possível estabelecer os valores de posição da partícula, convertendo o valor de um espaço de busca quântico para um resultado baseado em um espaço de soluções Newtoniano [Xi et al. 2007, Sun et al. 2004*a*]. A partir da Equação 2.27 (densidade de probabilidade), utiliza-se o método de Monte Carlo [8] para se estabeleçer a atualização de posição (Equação 2.35).

$$X_i(t+1) = \begin{cases} LIP_i + \frac{L_i(t)}{2} \ln(\frac{1}{R_{(u)}}) & \text{se } rand < 0.5 \\ LIP_i - \frac{L_i(t)}{2} \ln(\frac{1}{R_{(u)}}) & \text{se } rand > 0.5 \end{cases} \tag{2.35}$$

Substituindo a variável $L_i(t)$, a Equação 2.36 mostra como ocorre a atualização de posição do algoritmo QPSO.

$$X_i(t+1) = \begin{cases} LIP_i + \beta \left| P_{M(i)} - X_i(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand < 0.5 \\ LIP_i - \beta \left| P_{M(i)} - X_i(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand > 0.5 \end{cases} \tag{2.36}$$

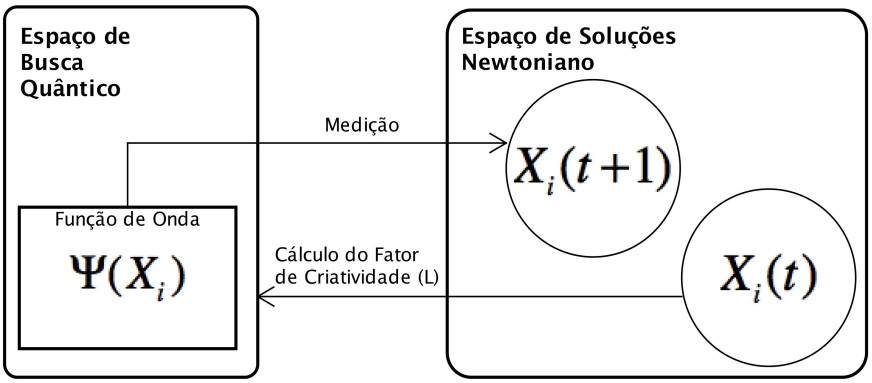
As variáveis descritas nas Equações 2.35 e 2.36 são: Parâmetro de criatividade da partícula i ($L_i(t)$); Fator de contração-expansão (β); Ponto de inclinação (*Learning Inclination Point*) (LIP_i); Valor atual de posição da partícula i ($X_i(t)$); Soma da média dos melhores locais (P_{Me}).; Ponto de inclinação (*Learning Inclination Point*) (LIP_i); Número aleatório gerado uniformemente no intervalo de 0 até 1 ($R_{(u)}$).

[8] Segundo [Hastings 1970] o método de Monte Carlo consiste em quaisquer métodos estatísticos baseados em grandes quantidades de amostragens aleatórias com o objetivo de extrair resultados numéricos, onde repetições sucessivas de simulações são realizadas para calcular probabilidades heuristicamente

Ilustração de Movimento e Pseudo-Código

Dada a natureza dos fenômenos quânticos, o espaço de busca e o espaço de solução são diferentes. Segundo [Herrera 2007] a função de onda descreve o estado da partícula em um espaço de busca quântico (Equação 2.25), não provendo informações acerca da posição da partícula. Nesse cenário, é necessária a conversão entre o espaço de busca quântico para o espaço de soluções Newtoniana (clássico). Isto é feito através do colapso da função de onda, obtendo assim a medida de posição da partícula. A Figura 2.8 mostra a atualização de posição envolvendo os espaços de busca e de soluções no QPSO.

Figura 2.8: Ilustração do espaço de busca (PSO e QPSO). Adaptado de:[Sun et al. 2004*a*]



O Algoritmo 3 demonstra os procedimentos do QPSO com fator α.

Algoritmo 3: QPSO com fator α

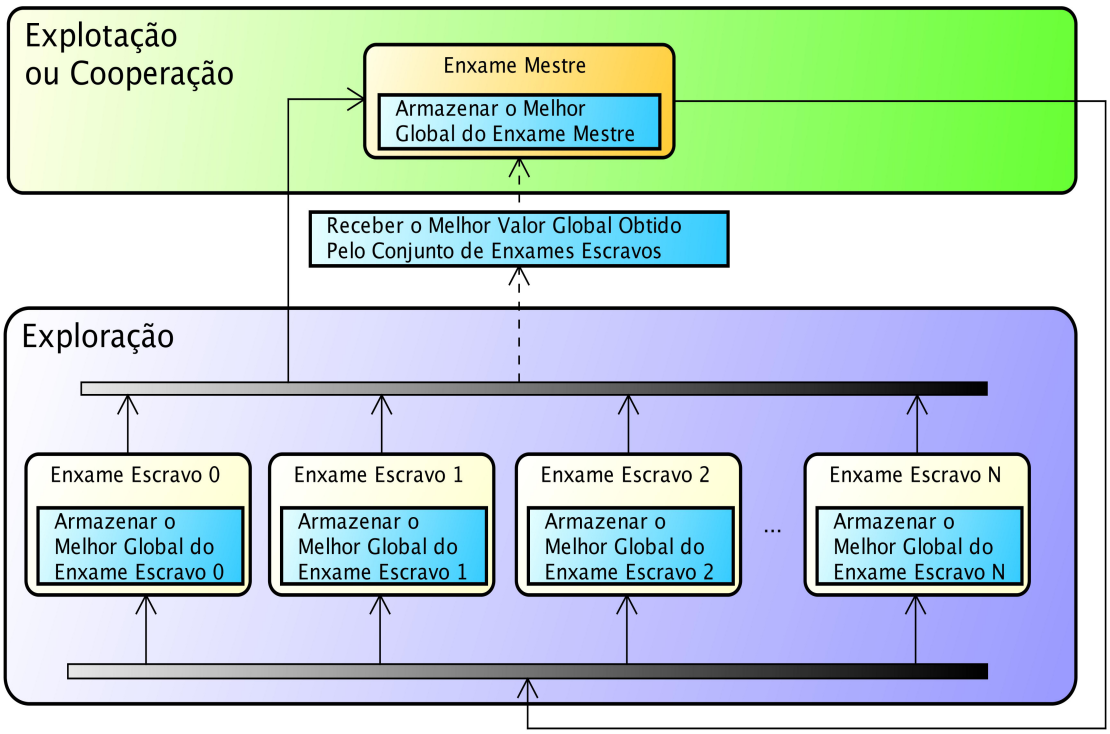
```
Alocar memória para estruturas do enxame (posição (X));
Alocar memória para melhores valores do enxame (local (P_i), global (P_g));
for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
    for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
        Inicializar os valores de posição;
    Avaliar partícula (Fitness);
    Inicializar os melhor local da partícula (P_i);
    Obter o melhor global do enxame (P_g);
for iter ← 1 to (QUANTIDADE DE ITERAÇÕES−1 do
    Atualizar fator β (Equação 2.32 ou 2.33);
    Atualizar fator α (Equação 2.29 ou 2.30);
    Obter a média dos melhores locais do enxame (P_M(i)) (Equação 2.31);
    for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
        for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
            Atualizar valor de ponto de inclinação (LIP ou p) (Equação 2.28);
            Atualizar valor de posição da variável j (Equação 2.36);
            Aplicar condições de contorno "Damping" aos valores de posição (Equação 2.11);
        Avaliar partícula (Fitness);
        Obter os melhor local da partícula (P_i);
        Obter o melhor global do enxame (P_g);
```

2.5 *Multi-Swarm Optimization*

A organização do ambiente multi-enxame utilizada nos algoritmos CEMSO, CQEMSO, COMSO, COEMSO e COQMSO é inspirada no modelo proposto por [Niu et al. 2005, Niu et al. 2007], denominado de MCPSO (*Multi-Populational Cooperative Particle Swarm Optimization*). Essa abordagem é organizada de modo que os enxames escravos sejam executados de forma paralela e/ou distribuída, sem trocar informações entre si, onde após a execução, os melhores resultados obtidos (melhores globais) serão transferidos para o enxame mestre. A cooperatividade de dados é aplicada somente no relacionamento entre o enxame mestre e os enxames escravos.

Após a execução paralela dos enxames escravos, o enxame mestre atualiza suas partículas com base nos novos valores de melhores globais de cada enxame escravo, bem como o melhor valor obtido por todos eles ($P_{g(S)}$) [El-Abd & Kamel 2008, Niu et al. 2007]. A Figura 2.9 demonstra o esquema de comunicação dos enxames escravos com o enxame mestre utilizados nos algoritmos CEMSO, CQEMSO, COMSO, COEMSO e COQMSO.

Figura 2.9: Esquema ilustrativo do modelo mestre-escravo. Adaptado de [Niu et al. 2007]



As operações de otimização realizadas nos enxames escravos e mestre, bem como as abordagens de interação entre os dois tipos de enxames são [Niu et al. 2007]:

- ***Relação "exploração-cooperação" vs "exploração-explotação"***: As interações entre o enxame mestre e os enxames escravos influenciam o equilíbrio entre a exploração e explotação, além de manterem uma diversidade adequada da população, mesmo quando ela está se aproximando da solução global, reduzindo assim o risco de convergir apenas para ótimos locais. As relações entre os enxames mestre e escravos são classificadas através de duas perspectivas:
  1. ***Abordagem cooperativa***: Este esquema atua sob um cenário sinergístico, onde o enxame mestre atualiza suas partículas levando em consideração a melhor solução obtida pelos enxames escravos, ambos com a mesma relevância;
  2. ***Abordagem competitiva***: Este esquema é baseado em um cenário antagônico e de caráter exploratório, onde o enxame mestre se utiliza da melhor solução obtida pelos enxames escravos no intuito de aprimorar suas partículas. Compete diretamente com os enxames escravos na busca da melhor solução.
- ***Fator de interação social aplicado ao melhor resultado encontrado pelos enxames escravos (C_3)***: O enxame mestre atua de modo que a melhor partícula encontrada pelos enxames escravos exerça influência na orientação no mesmo, independente da abordagem utilizada. Para que isso seja possível, [Niu et al. 2005] introduziu um terceiro parâmentro de interação na equação de atualização de velocidade.

As Equações 2.37, 2.38 e 2.39 descrevem a atualização de velocidade para o enxame mestre com abordagem competitiva, fator de migração e abordagem cooperativa, respectivamente.

$$V_{i(M)}(t+1) = wV_{i(M)}(t) + R_1C_1(P_i^M - X_{i(M)}(t)) + \Phi R_2C_2(P_{g(M)} - X_{i(M)}(t)) + (1-\Phi)R_3C_3(P_{g(S)} - X_{i(M)}(t)) \quad (2.37)$$

$$\Phi = \begin{cases} 0 & \text{se } P_{g(S)} < P_{g(M)} \\ 0.5 & \text{se } P_{g(S)} = P_{g(M)} \\ 1 & \text{se } P_{g(S)} > P_{g(M)} \end{cases} \quad (2.38)$$

$$V_{i(M)}(t+1) = wV_{i(M)}(t) + R_1C_1(P_{i(M)} - X_{i(M)}(t)) + R_2C_2(P_{g(M)} - X_{i(M)}(t)) + R_3C_3(P_{g(S)} - X_{i(M)}(t)) \quad (2.39)$$

As variáveis descritas nas Equações 2.37 e 2.38 e 2.39 são: Tempo atual da partícula (t); Fator de inércia (w); Valor atual de velocidade para partícula i ($V_i(t)$); Novo valor de velocidade para partícula i ($V_i(t+1)$); Valor atual de posição da partícula i ($X_i(t)$); Melhor solução encontrada pela partícula do enxame mestre (mínimo/máximo local) ($P_{i(M)}$); Melhor solução encontrada pelo enxame mestre (mínimo/máximo global) ($P_{g(M)}$); Melhor solução encontrada pelos enxames escravos (mínimo/máximo global) ($P_{g(S)}$); Fator de correção individual; Fator de correção individual (C_1); Fator de interação social do enxame mestre (C_2); Fator de interação social atribuído a melhor solução encontrada pelos enxames escravos (C_3); Número aleatórios gerados uniformemente entre de 0 até 1 (R_1, R_2 *e* R_3); Fator de migração (Φ);

Com base nos resultados obtidos por [Niu et al. 2005, Niu et al. 2007], além de testes efetuados no desenvolvimento deste trabalho, os algoritmos CEMSO, CQEMSO e versões multi-enxame dos algoritmos PSO (COMSO), EPSO (COEMSO) e QPSO (COQMSO) utilizam apenas a abordagem competitiva (Equações 2.37 e 2.38).

Capítulo 3

Algoritmos CEMSO e CQEMSO

Este capítulo tem o objetivo de mostrar as principais características dos algoritmos CEMSO (*Competitive Evolutionary Multi Swarm Optimization*) e CQEMSO (*Competitive Quantum-Behaviour Evolutionary Multi Swarm Optimization*). Além disso, apresenta-se o desenvolvimento dos algoritmos PSO+EE (*Particle Swarm Optimization with Evolutionary Estrategies*) e QPSO+EE (*Quantum-Behaviour Particle Swarm Optimization with Evolutionary Estrategies*). Dentre os tópicos abordados, destacam-se a apresentação uma topologia multi-enxame modificada de abordagem mestre-escravos, distribuição de partículas e enxames para execução na arquitetura CUDA, organização dos enxames escravos, uso das estratégias evolutivas no enxame mestre, dentre outras informações relevantes.

Encontra-se neste capítulo, a lista de trabalhos relacionados com os campos de estudo abordados nesta dissertação: Otimização por enxame de partículas clássico e quântico, estratégias evolutivas e algoritmo EPSO. Destacam-se os algoritmos bioinspirados implementados sob o paradigma GPGPU e que serviram de fonte de pesquisa para o desenvolvimento dos algoritmos PSO+EE, QPSO+EE, CEMSO e CQEMSO.

3.1 Apresentação e Trabalhos Relacionados

Para tornar os algoritmos PSO, QPSO e EPSO mais eficientes em termos de busca e otimização, os algoritmos CEMSO e CQEMSO propostos neste documento utilizam um ambiente multi-população baseado em uma topologia mestre-escravos. Com isso, busca-se concorrência entre os enxames escravos para proporcionar o melhor resultado possível ao enxame mestre. Além disso, as EE encontradas no EPSO são combinadas com o PSO clássico e QPSO com mecanismos de condições de contorno modificados para um melhor aproveitamento do espaço de busca.

Os algoritmos CEMSO e CQEMSO foram concebidos com base na observação das vantagens e limitações encontradas em pontos fundamentais nos algoritmos PSO, EPSO e QPSO, bem como nos mecanismos e técnicas trabalhadas nas implementações expostas neste documento.

Com base nos resultados obtidos por [Miranda & Fonseca 2002, Leite et al. 2010] pelo algoritmo EPSO, pode-se constatar o alto grau de eficiência das estratégias evolutivas aplicadas em um sistema baseado em enxame de partículas, onde além do processo de otimização modificado com base no algoritmo PSO, as réplicas mutadas oferecem um melhor aproveitamento na busca em cada iteração. Entretanto, ao não utilizar as constantes C_1 e C_2 pré-estabelecidas pelo usuário na atualização de velocidade da partícula original (vide Equação 2.21) e optando por uma abordagem baseada em valores gerados por operações gaussianas dos fatores individual e social para as réplicas, o algoritmo tende a perder a eficiência na busca nas últimas iterações. A medida em que as réplicas não encontram mais um resultado melhor em relação a partícula original, os valores de $w_{(i)}$ (fator de inércia), $C_{1(i)}$ (interação individual) e $C_{2(i)}$ (interação social) se mantém estáticos, aumentando as chances de estagnação em melhores locais.

Apesar do algoritmo QPSO já estar em um estágio avançado de desenvolvimento, com diversos trabalhos publicados que demonstram sua qualidade [e.g. Xi et al. 2007, Mani & Patvardhan 2010, Sun et al. 2005, Liu et al. 2008], a média e a variância obtida nos testes envolvendo três dos quatros os problemas de engenharia utilizados para a validação (DPV, MWTCS e SRD) apresentaram valores altos (os experimentos podem ser vistos no Capítulo 4). Um dos trabalhos envolvendo o uso de algoritmos com inspiração quântica (base para o QPSO) em problemas de engenharia pode ser encontrado em [Mani & Patvardhan 2010]. É importante ressaltar que durante a pesquisa efetuada para o desenvolvimento deste trabalho, não foram encontradas implementações do algoritmo QPSO utilizando estratégias evolutivas.

Recurso importante nos algoritmos PSO e derivados, as condições de contorno para posição e velocidade com base no modelo "*Damping*", apesar de demonstrar grande performance nas implementações do algoritmo PSO [e.g. Souza et al. 2013, Xu & Rahmat-Samii 2007], o procedimento reposiciona as partículas apenas ao limite mínimo ou máximo. Isso pode impactar na qualidade do processo de otimização e no aproveitamento do espaço de busca e soluções, caso uma quantidade considerável de partículas ultrapasse os limites pré-estabelecidos;

O uso de abordagens multi-enxame (ou multi-populacional) do PSO, EPSO e QPSO, mesmo com diversos trabalhos publicados reforçando a qualidade das soluções [e.g. den Bergh & Engelbrecht 2004, Niu et al. 2005, Mori & Okawa 2009, El-Abd & Kamel 2008, Veronese & Krohling 2009], possui poucas soluções relacionadas ao uso do paradigma GPGPU, bem como uma implementação voltada para o uso na arquitetura CUDA [e.g. Souza et al. 2013, Souza et al. 2014, Solomon et al. 2011].

Dada as limitações expostas, soma-se a elas a necessidade de produzir uma solução de busca que apresente resultados de qualidade no que tange problemas de otimização restritiva, os algoritmos CEMSO e CQEMSO tem como principais características e contribuiçoes, os seguintes items:

- ***Implementação de uma versão modificada da abordagem multi-enxame de topologia mestre-escravos proposta por [Niu et al. 2005, Niu et al. 2007] sob a arquitetura CUDA***: Os algoritmos CEMSO e CQEMSO utilizam recursos disponíveis na arquitetura CUDA, com o objetivo de proporcionar uma solução massivamente paralela de um ambiente multi-enxame, além de um maior desempenho no processo de busca e otimização com um tempo de execução viável para ambientes reais;
- ***Implementação das Estratégias Evolutivas de [Miranda & Fonseca 2002] no PSO clássico (PSO+EE)***: As estratégias evolutivas encontradas no algoritmo EPSO são utilizadas no PSO clássico, reinserindo a equação de atualização de velocidade 2.6 para as partículas originais;
- ***Introdução de uma nova metaheurística, que agrega as Estratégias Evolutivas de [Miranda & Fonseca 2002] ao QPSO (QPSO+EE)***: Assim como o algoritmo PSO+EE, o enxame de partículas de inspiração quântica é implementado com um mecanismo de estratégias evolutivas;
- ***Inclusão de uma nova abordagem de condição de contorno (Equivalência + Espelhamento + Damping)***: Além do procedimento de realocação de partículas "*Damping*" (Equações 2.10 e 2.11), é utilizado um procedimento de equivalência e espelhamento de partículas, visando uma exploração maior do espaço de busca, destacando pontos próximos dos limites mínimos e máximos;

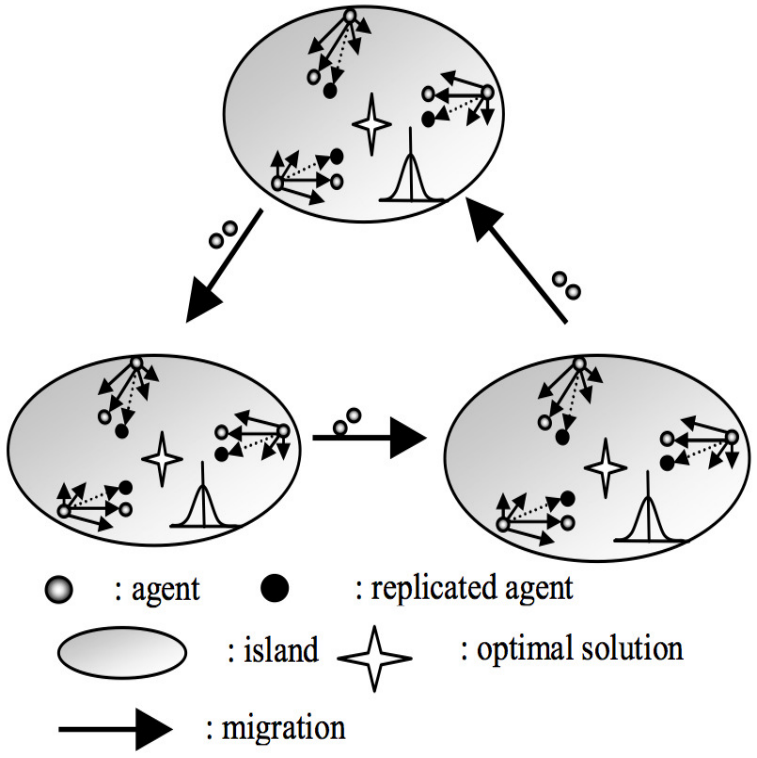
Os próximos tópicos abordam detalhadamente os pontos destacados anteriormente, bem como outras informações relevantes acerca dos algoritmos-base (PSO, EPSO, QPSO, PSO+EE, QPSO+EE) e de suas versões multi-enxame (COMSO (*Classic Competitive Multi Swarm Optimization*), COEMSO (*Classic Competitive Evolutionary Multi Swarm Optimization*), COQMSO (*Classic Competitive Quantum Multi Swarm Optimization*), CEMSO e CQEMSO).

3.1.1 Exemplos de Algoritmos Bioinspirada sob a Arquitetura CUDA

Esta subseção apresenta exemplos de trabalhos publicados envolvendo algoritmos bioinspirados (com destaque para a otimização por enxame de partículas) sob a arquitetura CUDA, que serviram de base para o desenvolvimento e comparação dos algoritmos PSO+EE e QPSO+EE, bem como suas extensões multi-enxame CEMSO e CQEMSO. Em relação aos algoritmos EPSO e QPSO, foi realizado um levantamento bibliográfico em relação as publicações envolvendo tais algoritmos, porém, nenhum trabalho com relação ao uso dos métodos criados por [Miranda & Fonseca 2002] (EPSO) e [Sun et al. 2005] (QPSO) em GPUs sob a arquitetura CUDA foi encontrado.

No entanto, deve-se ressaltar o trabalho de [Mori & Okawa 2009], onde é implementado uma versão paralela e multipopulacional do algoritmo EPSO desenvolvido por [Miranda & Fonseca 2002]. Neste trabalho, os autores utilizam a abordagem *Island* usada nos algoritmos genéticos, com o objetivo de refinar o processo de busca através da troca de partículas entre os enxames. Os autores não especificam o tipo de arquitetura computacional utilizada para o processamento paralelo dos enxames. A Figura 3.1 mostra a organização dos enxames no contexto da abordagem *Island*.

Figura 3.1: Organização dos enxames no algoritmo EPSO paralelo com troca de dados através da abordagem *Island*. Fonte:[Mori & Okawa 2009]



***Swarm's Flight: Accelerating the Particles Using C-CUDA* [Veronese & Krohling 2009]**

Este trabalho foi um dos primeiros envolvendo o uso do PSO com GPUs baseado na arquitetura CUDA. Um dos principais objetivos do trabalho é o de expor, a princípio, como os algoritmos baseados no modelo de inteligência de enxame e de busca e otimização poderiam tirar proveito desta tecnologia em relação ao ganho de desempenho. A implementação descrita consiste em utilizar um *Grid* como uma representação do enxame como um todo, onde cada *thread* seria responsável por uma partícula. Ao ser comparado com uma implementação sequencial escrita em linguagem C e outra em MatLab, os dados obtidos nos testes mostraram que a velocidade de execução no algoritmo CUDA-C é consideravelmente superior. é importante ressaltar que embora os resultados tenham mostrado um desempenho superior em relação as versões sequenciais do PSO, as funções de fitness usadas são apenas da classe de problemas não restitivos o que pode impactar no desempenho, visto que os problemas restritivos requerem diversas instruções de condição (*if*, *else*) para atender as restrições e, dependendo do problema, mais de uma equação a ser executada. Outro problema constatado neste trabalho é que os autores não apresentam os resultados ou o valor obtido pelo programa ao final da execução.

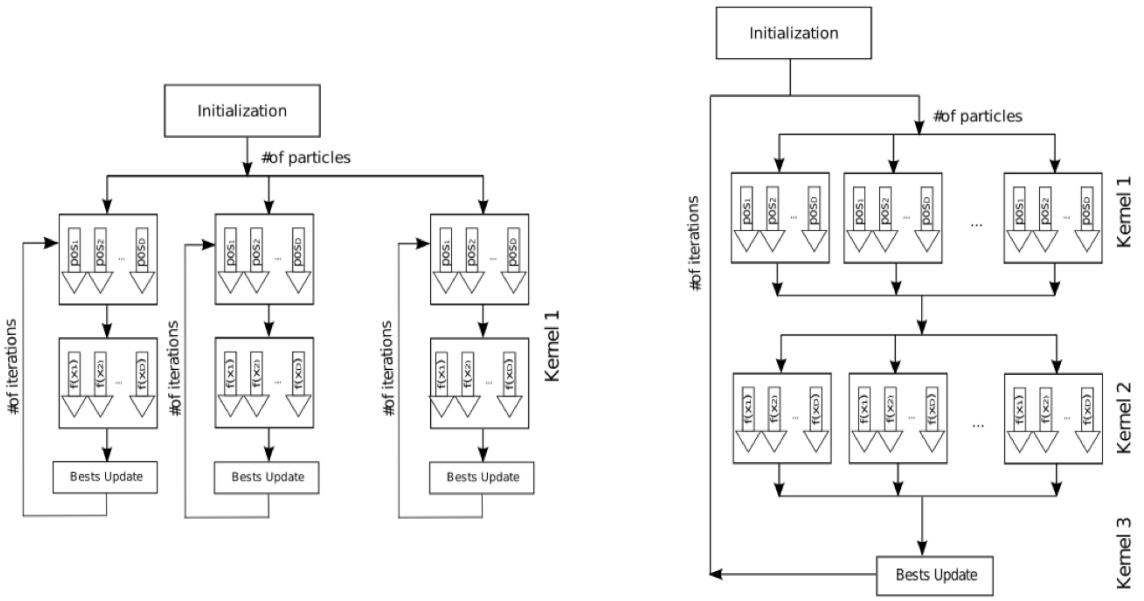
***Collaborative Multi-Swarm PSO for Task Matching Using Graphics Processing Units* [Solomon et al. 2011]**

Neste trabalho os autores expõem uma implementação do modelo cooperativo e multi-enxame do PSO executado sob a arquitetura CUDA para a resolução da tarefa de busca de padrões (*task matching*). Este algoritmo utiliza um tipo de distribuição de subenxames similar a encontrada em [den Bergh & Engelbrecht 2004] Uma característica que deve ser frisada neste trabalho é que os autores utilizam dados do tipo discreto e contínuo, sendo que todo o processo é realizado de forma contínua e ao final, os autores aplicam o arredondamento para valores discretos. Tal característica é importante, pois neste trabalho também são apresentados dados que mostram que uma implementação totalmente discreta do PSO é menos eficiente em relação a uma implementação contínua [Solomon et al. 2011]. Os resultados obtidos neste trabalham mostram melhorias consideráveis no tempo de execução e nos valores obtidos ao final da execução do processo de *task matching* quando comparados com um PSO de apenas um enxame. Porém, a qualidade da execução passa a se deteriorar a medida que a quantidade de dados para processamento começa a aumentar.

***GPU-based Asynchronous Particle Swarm Optimization* [Mussi et al. 2011]**

Este trabalho apresenta uma abordagem de execução assíncrona do PSO na GPU e expõe os problemas de uma implementação síncrona, afirmando que o processo sincronização de todas as *threads* acarretaria em perda de performance. Nessa implementação, os autores propõem um tratamento de dados diferente do apresentada no trabalho de [Veronese & Krohling 2009], onde cada *thread* representa um elemento específico da partícula, e as partículas são tratadas de em um laço de repetição. Os resultados obtidos, em relação a versão síncrona mostram um aumento de performance significativo. é importante ressaltar que na implementação síncrona exposta neste trabalho, são encontrados três *kernels* distintos, ou seja, temos três operações de sincronização de *threads*. Entretanto, assim como em [Veronese & Krohling 2009], este trabalho não contém testes realizados com funções restritivas. A Figura 3.2 mostra um esquema comparativo entre a versão síncrona e assíncrona.

Figura 3.2: Esquema comparativo do PSO síncrono (à direita) e assíncrono (à esquerda). Cada seta branca representa uma *thread* tratando um elemento da partícula e cada *block* uma partícula em uma iteração. Fonte:[Mussi et al. 2011]

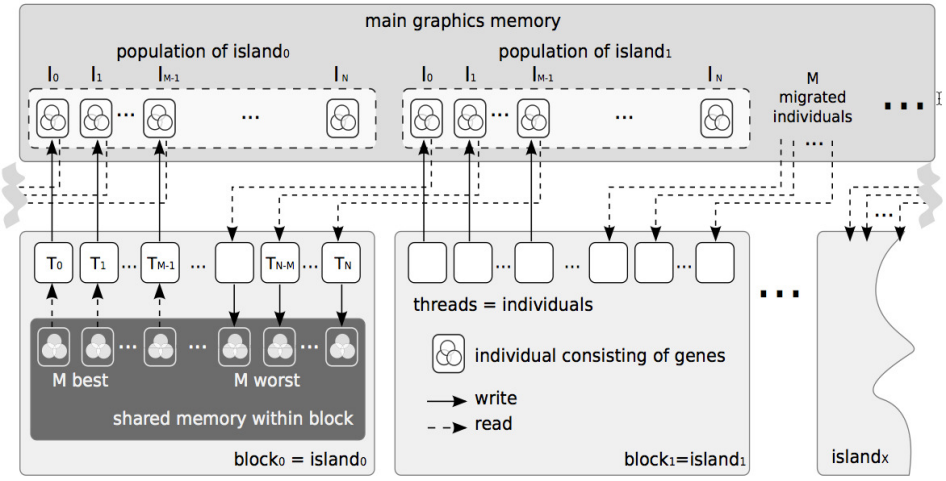


***Parallel Genetic Algorithm on the CUDA Architecture* [Petr et al. 2010]**

Com a adesão da arquitetura CUDA, os algoritmos bioinspirados, em especial os algoritmos genéticos, foram usados no intuito de otimizar o tempo de processamento e propor novos tipos de algoritmos. Sob o contexto do PSO multi-enxame, implementações do algoritmo genético utilizando a abordagem *Island* foram estudadas devido sua capacidade de troca de dados entre as populações, bem como o impacto desse mecanismo na busca e otimização.

Este artigo mostra uma implementação de um algoritmo genético paralelo através da modelo island em uma GPU. Esse tipo de abordagem, muito utilizada na literatura principalmente em ambiente de *clusters Beowulf* consiste na utilização de várias subpopulações isoladas que evoluem em paralelo e periodicamente trocam informações, migrando seus melhores indivíduos para subpopulações vizinhas [Linden 2008]. Os resultados apresentados pelos autores na comparação entre as implementações (CPU *single thread* e GPU) mostram um aumento na velocidade de execução de aproximadamente 7000 vezes no algoritmo executado na GPU, além de boas soluções obtidas ao final da execução. Embora os resultados apresentados sejam consideravelmente superiores, o artigo não expõe um algoritmo genético *Island* paralelo implementado sob um ambiente envolvendo CPUs (e.g. *cluster Beowulf*, ambientes de multiprocessamento simétrico) para comparação. A Figura 3.3 mostra o processo de migração de indivíduos entre as ilhas do AG *Island* sob a GPU.

Figura 3.3: Esquema do AG *island* implementado em GPU. Cada *thread* representa uma partícula, enquanto os *blocks* armazenam as populações (*islands*). Fonte:[Petr et al. 2010]

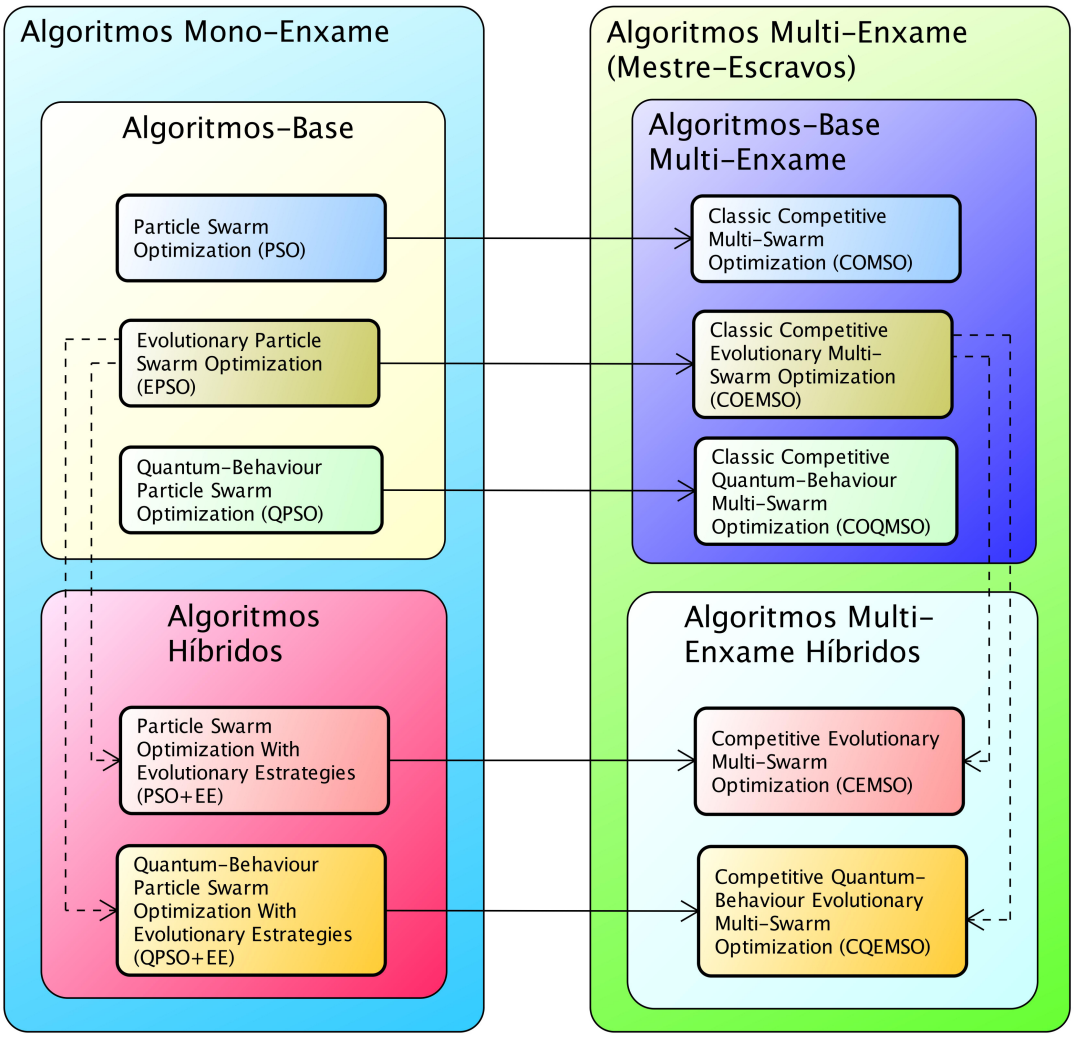


3.1.2 Breve Descrição dos Algoritmos Implementados

Durante o desenvolvimento deste trabalho, foram implementados, ao todo, dez metaheurísticas baseadas em enxame de partículas. Dessas, cinco são baseadas em otimização envolvendo apenas um enxame e cinco com abordagem multi-enxame de topologia mestre-escravos.

A Figura 3.4 mostra as relações entre os algoritmos de apenas um enxame (PSO, EPSO, QPSO, QPSO+EE, PSO+EE) e suas respectivas extensões multi-enxame com topologia mestre-escravos. As relações feitas com setas tracejadas indicam uma solução que recebe mecanismos de outros(as) metaheurísticas (e.g. PSO+EE e QPSO+EE recebendo as estratégias evolutivas do algoritmo EPSO), caracterizando um algoritmo híbrido.

Figura 3.4: Relação entre os algoritmos mono-enxame e multi-enxame



Algoritmos Desenvolvidos por Outros Autores

Segue abaixo a relação dos algoritmos publicados por outros autores e que foram utilizados como base para as soluções desenvolvidas nessa dissertação.

- ***PSO - Particle Swarm Optimization (Desenvolvido por [Kennedy & Eberhart 1995])***: Algoritmo de busca e otimização baseado na movimentação de enxames, tendo como regra de movimento a mecânica clássica. Detalhes podem ser encontrados no capítulo 2, subtópico 2.2;
- ***EPSO - Evolutionary Particle Swarm Optimization (Desenvolvido por [Miranda & Fonseca 2002])***: Algoritmo de busca e otimização baseado na movimentação de enxames com elementos de EE e procedimento de atualização de velocidade modificado para as partículas originais. Esta solução tem como regra de movimento a mecânica clássica e processos evolutivos. Detalhes podem ser encontrados no capítulo 2, subtópico 2.3;
- ***QPSO - Quantum-Behaviour Particle Swarm Optimization (Desenvolvido por [Sun et al. 2004*b*])***: Algoritmo de busca e otimização baseado na movimentação de enxames, tendo como regra de movimento a mecânica quântica. Detalhes podem ser encontrados no capítulo 2, subtópico 2.4;
- ***COMSO - Classic Competitive Multi-Swarm Optimization (Baseado no trabalho desenvolvido por [Niu et al. 2005, Niu et al. 2007])***: Algoritmo de busca e otimização multi-enxame através da topologia mestre-escravos inspirado no modelo de [Niu et al. 2005, Niu et al. 2007]. Utiliza como base o algoritmo PSO clássico. Detalhes do algoritmo podem ser vistos no capítulo 2, subtópico 2.5 e a implementação é descrita no anexo desta dissertação;

Algoritmos Desenvolvidos Neste Trabalho

Segue abaixo a relação dos algoritmos desenvolvidos nesta dissertação.

- ***PSO+EE - Particle Swarm Optimization With Evolutionary Estrategies***: Algoritmo de busca e otimização baseado na movimentação de enxames encontrada no PSO clássico (mantém a atualização de velocidade original) que adiciona as EE do EPSO e tem como regra de movimento a mecânica clássica. Detalhes podem ser encontrados no capítulo 3, subtópico 3.2;

- ***QPSO+EE - Quantum-Behaviour Particle Swarm Optimization With Evolutionary Estrategies***: Algoritmo de busca e otimização baseado na movimentação de enxames encontrada no QPSO clássico (mantém a atualização de posição original) que adiciona as EE do EPSO e tem como regra de movimento a mecânica clássica. Detalhes podem ser encontrados no capítulo 3, subtópico 3.3;
- ***COEMSO - Classic Competitive Evolutionary Multi-Swarm Optimization***: Algoritmo de busca e otimização multi-enxame através da topologia mestre-escravos inspirado no modelo de [Niu et al. 2005, Niu et al. 2007]. Utiliza como base o algoritmo EPSO, tendo como característica a inserção das EE tanto nos enxames escravos quanto no enxame mestre. Os principais mecanismos do algoritmo podem ser vistos no capítulo 3, subtópico 3.2 e a implementação é descrita no Anexo;
- ***COQMSO - Classic Competitive Quantum-Behaviour Multi-Swarm Optimization***: Algoritmo de busca e otimização multi-enxame através da topologia mestre-escravos inspirado no modelo de [Niu et al. 2005, Niu et al. 2007]. Utiliza como base o algoritmo QPSO clássico. Os principais mecanismos do algoritmo podem ser vistos no capítulo 3, subtópico 3.3 e a implementação é descrita no anexo desta dissertação;
- ***CEMSO - Competitive Evolutionary Multi-Swarm Optimization***: Algoritmo de busca e otimização multi-enxame através da topologia mestre-escravos inspirado no modelo de [Niu et al. 2005, Niu et al. 2007]. Utiliza como base o algoritmo PSO+EE, tendo como características a utilização do procedimento de atualização de velocidade do PSO clássico para as partículas originais e a inserção das estratégias evolutivas tanto nos enxames escravos quanto no enxame mestre. Os principais mecanismos do algoritmo podem ser vistos no capítulo 3, subtópico 3.2 e a implementação é descrita nos subtópicos 3.5 e 3.7;
- ***CQEMSO - Competitive Quantum-Behaviour Multi-Swarm Optimization With Evolutionary Estrategies***: Algoritmo de busca e otimização multi-enxame através da topologia mestre-escravos inspirado no modelo de [Niu et al. 2005, Niu et al. 2007]. Utiliza como base o algoritmo QPSO+EE, tendo como características a utilização do procdimento de atualização de posição do QPSO clássico e a inserção das estratégias evolutivas tanto nos enxames escravos quanto no enxame mestre. Os principais mecanismos do algoritmo podem ser vistos no capítulo 3, subtópico 3.4 e a implementação é descrita no subtópico 3.5 e 3.10;

3.1.3 Análise Comparativa sobre os Trabalhos Relacionados e os Algoritmos Implementados

Os trabalhos descritos nos subtópicos anteriores serviram como base inicial para o estudo de uma abordagem multi-enxame dos algoritmos PSO, EPSO e QPSO. É importante ressaltar que os *insights* obtidos por tais trabalhos foram de grande importância no desenvolvimento de mecanismos customizados e posteriormente combinados com as soluções propostas por [Niu et al. 2007]. A Tabela 3.1 mostra algumas das características das soluções relacionadas e dos algoritmos PSO+EE, QPSO+EE, CEMSO e CQEMSO.

Segue-se os algoritmos usados para comparação na Tabela 3.1:

- ***Alg. 1 [Mussi et al. 2011]***: GPU-based Asynchronous Particle Swarm Optimization;
- ***Alg. 2 [Veronese & Krohling 2009]***: Swarm's Flight: Accelerating the Particles Using C-CUDA;
- ***Alg. 3 [Solomon et al. 2011]***: Collaborative Multi-Swarm PSO for Task Matching Using Graphics Processing Units;
- ***Alg. 4 [Petr et al. 2010]***: Parallel Genetic Algorithm on the CUDA Architecture;
- ***Alg. 5 [Niu et al. 2007]***: MCPSO: A Multi-Swarm Cooperative Particle Swarm Optimizer;
- ***Alg. 6 [Mori & Okawa 2009]***: Integration of Parallel EPSO and Variable TS for Unit Commitment with Nonsmooth Fuel Cost Functions;

Segue-se os critérios de comparação usados na Tabela 3.1:

- ***Crit. 1***: Ambiente multi-população;
- ***Crit. 2***: Troca de dados entre populações;
- ***Crit. 3***: Topologia mestre-escravos;
- ***Crit. 4***: Abordagem *Island*;
- ***Crit. 5***: Possui implementação na arquitetura CUDA publicada;
- ***Crit. 6***: Mecanismo de otimização (***O*** para versão original, ***H*** para método híbrido e ***M*** para abordagem modificada em relação a versão original);
- ***Crit. 7***: Condições de contorno;
- ***Crit. 8***: Utiliza Estratégias Evolutivas (EE);

Tabela 3.1: Diferenças entre os algoritmos QPSO e QPSO+EE

Algoritmo	Crit. 1	Crit. 2	Crit. 3	Crit. 4	Crit. 5	Crit. 6	Crit. 7	Crit. 8
PSO original	-	-	-	-	X	O	X	-
EPSO original	-	-	-	-	X[1]	M	N/A	X
QPSO original	-	-	-	-	X[1]	O	N/A	-
PSO+EE	-	-	-	-	X[1]	H	N/A	X
QPSO+EE	-	-	-	-	X[1]	H	N/A	X
COMSO	X	X[2]	X	-	-	O[3]	X	-
COEMSO	X	X[2]	X	-	X[1]	M[3]	X	X
COQMSO	X	X[2]	X	-	-	O[3]	X	-
CEMSO	X	X[2]	X	-	X	H[3]	X	X
CQEMSO	X	X[2]	X	-	X	H[3]	X	X
Alg. 1	-	-	-	-	X	O	N/A	-
Alg. 2	-	-	-	-	X	O	N/A	-
Alg. 3	X	X	-	-	X	O	N/A	-
Alg. 4	X	X	-	X	X	O	N/A	X[5]
Alg. 5	X	X[2]	X	-	-	O[3]	N/A	-
Alg. 6	X	X	-	X	-	H[4]	N/A	X

Os pontos destacados na Tabela 3.1 são detalhados abaixo:

1. Com base na pesquisa bibliográfica realizada, os trabalhos publicados que utilizam a arquitetura CUDA foram submetidos pelo autor desta dissertação [Souza et al. 2013, Souza et al. 2014];
2. A troca de dados entre as populações ocorre apenas entre o enxame mestre e os enxames escravos. Não ocorre troca de informações entre os enxames escravos;
3. Os mecanismos de otimização deste ponto são inseridos no contexto do enxame mestre, sofrendo modificações na estrutura, porém, mantendo a base intacta;
4. O algoritmo desenvolvido por [Mori & Okawa 2009] consiste em uma versão modificada do algoritmo EPSO, onde inclui-se no processo de otimização, mecanismos de busca tabu entre as populações;
5. Embora os algoritmos genéticos (AG) utilizem uma abordagem evolucionária para o processo de busca e otimização, tais mecanismos diferem das soluções de estratégias evolutivas (EE) desenvolvidas por [Schwefel 1965, Rechenberg 1973, Schwefel & Rudolph 1995]. No AG, a criação de descendentes (novas soluções) é baseada no operador de recombinação, enquanto que nas EE, as novas soluções são geradas a partir dos operadores de mutação, gerando assim, um novo indivíduo;

A partir do tópico 3.2, são apresentados os detalhes das implementações propostas nesta dissertação: PSO+EE, QPSO+EE, CEMSO e CQEMSO.

3.2 Estratégias Evolutivas e PSO Clássico (PSO+EE)

O algoritmo de otimização por enxame de partículas clássico com estratégias evolutivas surge como uma versão aprimorada do algoritmo PSO, onde são atribuídos conceitos referentes às estratégias evolutivas propostas por [Miranda & Fonseca 2002] agregados a uma versão clássica do algoritmo PSO com fator de inércia. Diferentemente do algoritmo EPSO, que utiliza as estratégias evolutivas como motor principal do mecanismo de busca, onde os fatores de inércia, de interação individual e social das partículas originais dependem diretamente do processo de mutação de $C_{1(i)}$, $C_{2(i)}$ e $w_{(i)}$, além de novas e melhores posições geradas pelas réplicas, o algoritmo PSO+EE utiliza as réplicas mutadas como um processo de busca auxiliar, afim de explorar novas posições de maneira independente. As principais diferenças entre o algoritmo PSO clássico, PSO+EE e EPSO podem ser constatadas na Tabela 3.2.

Tabela 3.2: Diferenças entre os algoritmos PSO, PSO+EE e EPSO

Algoritmo	PSO	PSO+EE	EPSO
Versão Multi-Enxame	COMSO	CEMSO	COEMSO
Utilização das EE	Não	Sim	Sim
Atualização de Velocidade (Original)	Equação 2.5	Equação 2.5	Equação 2.22
Parâmetros C_1, C_2	Constante	Constante e Variável[1]	Variável
Fator de Inércia (w)	Variável[2]	Variável[3]	Variável
Configuração de C_1 e C_2	Usuário	Usuário e Estocástico[1]	Estocástico
Limites de w	W_{MIN} e W_{MAX} [4]	W_{MIN} e W_{MAX} [4,5]	Sem Limites[5]

1. Os valores de C_1 e C_2 são constantes no que tange a atualização de velocidade da partícula original, sendo pré-configuradas pelo usuário. Para as réplicas, os valores de C_1 e C_2 são utilizados como base para o processo de mutação, consequentemente gerando novos parâmetros ($mC_{1(i)}$ e $mC_{2(i)}$);
2. Os valores de w podem adquirir um comportamento de decrescimento linear ou randômico (vide Equações 2.1 e 2.2);
3. Os valores de w adquirem o mesmo comportamento descrito no item 2, com a diferença de que as réplicas terão valor do fator de inércia (w) submetidos ao processo de mutação, gerando assim novos valores de inércia ($mw_{(i)}$);
4. Os limites de w dependem da equação de atualização do fator de inércia utilizada. No caso da Equação 2.2, os valores de W_{MIN} e W_{MAX} serão 0 e 0.5;
5. Apesar de [Miranda & Fonseca 2002] não definir limites mínimos e máximos para o fator de inércia aplicado as réplicas, os valores são inicializados através de um gerador de números aleatórios uniforme em um intervalo de zero a um.

Assim como o algoritmo EPSO, a partícula original gera uma quantidade determinada de réplicas. Entretanto, o algoritmo PSO+EE sempre irá gerar novos valores de interação individual, social e fator de inércia para as réplicas tendo por base as constantes pré-estabelecidas pelo usuário. As Equações 3.1, 3.2 e 3.3 mostram o processo de mutação modificado para os parâmetros $mC_{1(i)}$ (fator de inércia), $mC_{2(i)}$ (interação individual:) e $mw_{(i)}$ (interação social) utilizados pelo algoritmo PSO+EE.

$$mw_{(i)} = w + (1 + \sigma N(0,1)) \tag{3.1}$$

$$mC_{1(i)} = C_1 + (1 + \sigma N(0,1)) \tag{3.2}$$

$$mC_{2(i)} = C_2 + (1 + \sigma N(0,1)) \tag{3.3}$$

As réplicas tem por finalidade atuar apenas como um processo de busca secundário baseado em valores mutados de C_1, C_2 e w, onde as constantes de interação individual e social configuradas pelo usuário com base no problema a ser solucionado oferecem valores mais adequados para o processo de mutação. De uma maneira simplificada, pode-se afirmar que, semelhantemente ao PSO clássico, a performance do algoritmo PSO+EE pode variar de acordo com os valores estabelecidos pelas constantes C_1 e C_2, apesar de que, com a utilização de estratégias evolutivas, projeta-se que a velocidade de convergência seja significativamente maior e com um melhor aproveitamento do espaço de busca, tanto pela partícula original, quanto pelas suas respectivas réplicas.

Ao reinserir a equação de atualização de velocidade do PSO clássico para as partículas originais (Equação 2.5), o algoritmo PSO+EE não dependerá dos novos valores $mC_{1(i)}$, $mC_{2(i)}$ e $mw_{(i)}$ para manter o processo de otimização nas últimas iterações pois, caso as réplicas não gerem uma partícular melhor do que a original, a partícula original ainda contará com os valores de C_1 e C_2 originais sendo multiplicados pelos coeficientes R_1 e R_2, o que diminui a possibilidade de estagnação em um ótimo local.

Com as adaptações propostas pelo algoritmo PSO+EE, obtem-se uma solução computacional que agrega as vantagens encontradas tanto no PSO clássico quanto no EPSO em uma única solução, onde se pode destacar um melhor aproveitamento do espaço de busca pelas réplicas geradas (em relação ao EPSO) com uma configuração mais adequada das constantes de interação individual e social (em relação ao PSO clássico).

3.2.1 PSO+EE e EPSO Aplicados ao Enxame Mestre (CEMSO e COEMSO)

Assim como nos enxames escravos, os algoritmos PSO+EE e EPSO podem ser implementados em um enxame mestre. Utilizando a abordagem dos algoritmos CEMSO e COEMSO, a solução proposta apresenta modificações significativas, bem como novos mecanismos que auxiliam no processo de otimização do enxame mestre. Os principais recursos usados no PSO+EE e EPSO aplicados ao enxame mestre são:

- ***Processo de mutação aplicado ao fator de interação social dos enxames escravos***: Semelhantemente ao fatores de inércia ($mw_{(i)}$), interação individual ($mC_{1(i)}$) e social ($mC_{2(i)}$), o valor de interação social aplicado aos enxames escravos também é submetido ao processo de mutação, gerando um novo parâmetro chamado $mC_{3(i)}$;
- ***Pertubação do melhor global encontrado pelo enxame mestre (réplicas)***: Assim como na Equação 2.20, os valores de posição de $P_{g(M)}$ são submetidos a um processo de pertubação no intuito de gerar novos referenciais para o processo de busca das réplicas;
- ***Novas equações de atualização de velocidade para as réplicas***: A adição das estratégias evolutivas no enxame mestre utiliza uma versão modificada da Equação 2.22 implementada com base nas equações propostas por [Niu et al. 2005] (2.37 e 2.39), onde o fator de interação social aplicado aos enxames escravos (C_3 para o PSO/PSO+EE e $C_{3(i)}$ para o algoritmo EPSO) é submetido a um processo de mutação, gerando novos parâmetros para as réplicas.

As Equações 3.4, 3.5 e 3.6 demonstram o procedimento de mutação do fator de interação social aplicado aos enxames escravos (C_3 para o algoritmo EPSO (COEMSO) e $C_{3(i)}$ para o algoritmo PSO+EE (CEMSO)), bem como o processo de pertubação do melhor global encontrado pelo enxame mestre ($mP_{g(M)}$) para os algoritmos PSO+EE (CEMSO) e EPSO (COEMSO).

$$mC_{3(i)} = C_{3(i)} + (1 + \sigma N(0,1)) \tag{3.4}$$

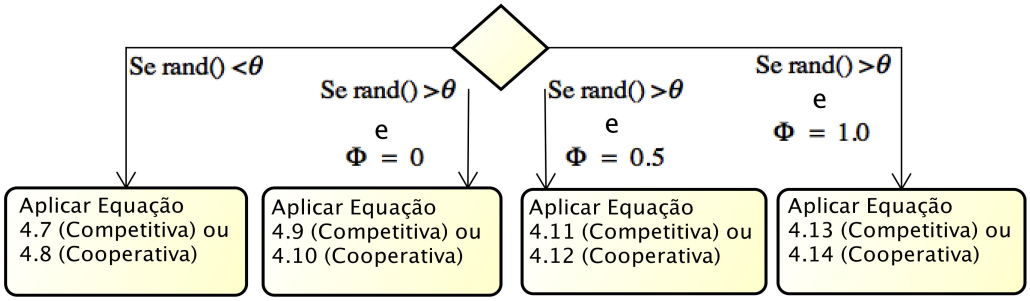
$$mC_{3(i)} = C_3 + (1 + \sigma N(0,1)) \tag{3.5}$$

$$mP_{g(M)} = P_{g(M)} + (1 + \omega_i N(0,1)) \tag{3.6}$$

Atualização de Velocidade

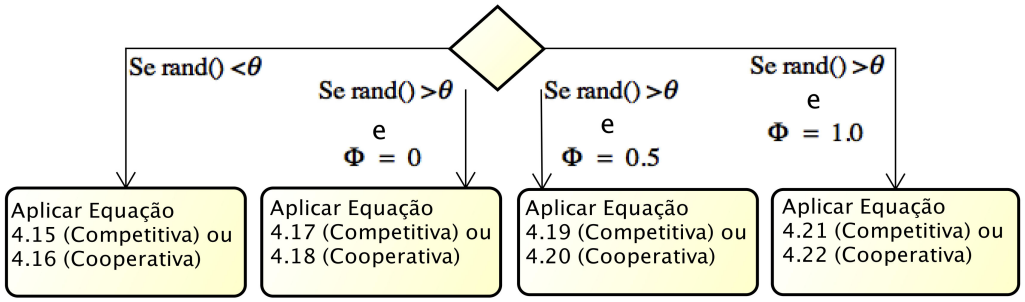
O cálculo de atualização de velocidade para as partículas originais do enxame mestre no algoritmo EPSO consiste na inserção dos fatores de interação individual e social dos enxames mestre e escravos nas equações proposta por [Niu et al. 2005] (2.37 e 2.39). A Figura 3.5 demonstra o processo de escolha da equação para o cálculo de velocidade das partículas originais do enxame mestre no algoritmo COEMSO (EPSO).

Figura 3.5: Seleção da equação de velocidade para as partículas originais



Para o cálculo de velocidade das réplicas no enxame mestre, foram adicionadas duas abordagens distintas para a geração de novos valores, onde são utilizados para seleção do procedimento de atualização, tanto o valor da constante θ quanto o fator de migração Φ. No caso do fator de migração, além de ser utilizado para dar maior relevância ao melhor valor global obtido em todo processo de busca ($P_{g(S)}$ ou $P_{g(M)}$), o mesmo atua como parâmetro de escolha para três equações de atualização de velocidade. A Figura 3.6 demonstra a seleção da equação para o cálculo de velocidade das réplicas (CEMSO e COEMSO).

Figura 3.6: Seleção da equação de velocidade para as réplicas



Atualização de Velocidade - Partículas Originais

As equações para as partículas originais expostas na Figura 3.5 são exibidas abaixo.

1. Caso $rand < \theta$, a nova velocidade será obtida através das Equações 3.9 (abordagem competitiva) e 3.10 (abordagem cooperativa). As principais características são:

   - Os três componentes de referência para a atulização estão presentes: Melhor local, melhor global do enxame mestre e melhor global dos enxames escravos;
   - Valor original do melhor global (enxame mestre) é submetido a pertubação;

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(mP_{i(M)} - X_{i(M)}(t)) + \Phi C_{2(i)}(P_{g(M)} - X_{i(M)}(t)) + (1-\Phi)C_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \quad (3.7)$$

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(mP_{i(M)} - X_{i(M)}(t)) + C_{2(i)}(P_{g(M)} - X_{i(M)}(t)) + C_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \quad (3.8)$$

2. Caso $rand > \theta$ e o valor do fator de migração Φ seja igual a zero, a nova velocidade será obtida através das Equações 3.9 (abordagem competitiva) e 3.10 (abordagem cooperativa). As principais alterações são:

   - Ausência dos componentes de interação social do melhor resultado obtido pelos enxames escravos ($C_{3(i)}$);
   - Valor do melhor global (enxame mestre) é mantido sem pertubação;

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + \Phi C_{2(i)}(P_{g(M)} - X_{i(M)}(t)) \quad (3.9)$$

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + C_{2(i)}(P_{g(M)} - X_{i(M)}(t)) \quad (3.10)$$

3. Caso *rand* $>\theta$ e o valor do fator de migração Φ seja igual a 0.5, a nova velocidade será obtida através das Equações 3.11 (abordagem competitiva) e 3.12 (abordagem cooperativa). As principais características são:

   - Os três componentes de referência para a atulização estão presentes: Melhor local, melhor global do enxame mestre e melhor global dos enxames escravos;
   - Valor do melhor global (enxame mestre) é mantido sem pertubação;

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + \Phi C_{2(i)}(P_{g(M)} - X_{i(M)}(t)) + (1-\Phi)C_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \quad (3.11)$$

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + C_{2(i)}(P_{g(M)} - X_{i(M)}(t)) + C_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \quad (3.12)$$

4. Caso *rand* $>\theta$ e o valor do fator de migração Φ seja igual a um, a nova velocidade será obtida através das Equações 3.13 (abordagem competitiva) e 3.14 (abordagem cooperativa). As principais características são:

   - Ausência dos componentes de interação social do melhor resultado obtido pelo enxame mestre ($C_{2(i)}$);
   - Valor do melhor global (enxame mestre) é mantido sem pertubação;

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + (1-\Phi)C_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \quad (3.13)$$

$$V_{i(M)}(t+1) = w_{(i)}V_{i(M)}(t) + C_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + C_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \quad (3.14)$$

No caso do algoritmo CEMSO (PSO+EE), a atualização de velocidade é calculada pelas equações originais do PSO e modificadas por [Niu et al. 2005] (2.37 e 2.39).

Atualização de Velocidade - Réplicas

As equações para as réplicas mencionadas na Figura 3.6 são definidas em sequência.

1. Caso $rand < \theta$, a nova velocidade será obtida através das Equações 3.15 (abordagem competitiva) e 3.16 (abordagem cooperativa). As principais características são:
   - Os três componentes de referência para a atulização estão presentes: Melhor local, melhor global do enxame mestre e melhor global dos enxames escravos;
   - Valor do melhor global (enxame mestre) é submetido a pertubação;

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + \Phi mC_{2(i)}(mP_{g(M)} - X_{i(M)}(t)) + (1-\Phi)mC_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \tag{3.15}$$

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + mC_{2(i)}(mP_{g(M)} - X_{i(M)}(t)) + mC_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \tag{3.16}$$

2. Caso $rand > \theta$ e o valor do fator de migração Φ seja igual a zero, a nova velocidade será obtida através das Equações 3.17 (abordagem competitiva) e 3.18 (abordagem cooperativa). As principais alterações são:
   - Ausência dos componentes de interação social do melhor resultado obtido pelos enxames escravos ($mC_{3(i)}$);
   - Valor do melhor global (enxame mestre) é mantido sem pertubação;

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + \Phi mC_{2(i)}(P_{g(M)} - X_{i(M)}(t)) \tag{3.17}$$

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + mC_{2(i)}(P_{g(M)} - X_{i(M)}(t)) \tag{3.18}$$

3. Caso $rand > \theta$ e o valor do fator de migração Φ seja igual a 0.5, a nova velocidade será obtida através das Equações 3.19 (abordagem competitiva) e 3.20 (abordagem cooperativa). As principais características são:

   - Os três componentes de referência para a atulização estão presentes: Melhor local, melhor global do enxame mestre e melhor global dos enxames escravos;
   - Valor do melhor global (enxame mestre) é mantido sem pertubação;

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + \Phi mC_{2(i)}(P_{g(M)} - X_{i(M)}(t)) + (1-\Phi)mC_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \tag{3.19}$$

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + mC_{2(i)}(P_{g(M)} - X_{i(M)}(t)) + mC_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \tag{3.20}$$

4. Caso $rand > \theta$ e o valor do fator de migração Φ seja igual a um, a nova velocidade será obtida através das Equações 3.21 (abordagem competitiva) e 3.22 (abordagem cooperativa). As principais características são:

   - Ausência dos componentes de interação social do melhor resultado obtido pelo enxame mestre ($mC_{2(i)}$);
   - Valor do melhor global (enxame mestre) é mantido sem pertubação;

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + (1-\Phi)mC_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \tag{3.21}$$

$$rV_{i(M)}(t+1) = mw_{(i)}V_{i(M)}(t) + mC_{1(i)}(P_{i(M)} - X_{i(M)}(t)) + mC_{3(i)}(P_{g(S)} - X_{i(M)}(t)) \tag{3.22}$$

3.3 Estratégias Evolutivas e QPSO (QPSO+EE)

O algoritmo de otimização por enxame de partículas com inspiração quântica e EE é uma nova metaheurística híbrida, em que os mecanismos de EE são adicionados ao processo de otimização do algoritmo QPSO com fator de decrescimento aplicado a média dos melhores locais (variável α). Assim como no algoritmo PSO+EE, a utilização das estratégias evolutivas no QPSO visa a inclusão de um motor de busca auxiliar na geração de réplicas com novos valores de posição baseadas nas partículas originais, otimizando a busca através da geração de novas soluções.

As principais modificações propostas para o algoritmo QPSO+EE incluem uma nova equação para o cálculo de *LIP* e fator de contração-expansão (β) submetido ao processo de mutação durante a geração das réplicas. As diferenças entre os algoritmos QPSO e QPSO+EE podem ser constatadas na Tabela 3.3.

Tabela 3.3: Diferenças entre os algoritmos QPSO e QPSO+EE

Algoritmo	QPSO	QPSO+EE
Versão Multi-Enxame	COQMSO	CQEMSO
Utilização das EE	Não	Sim
Atualização de Posição (Original)	Equação 2.34	Equação 2.34
Parâmetros R_{c1} e R_{c2}	Variável (Estocástico)	Variável (Estocástico)[1]
Fator de Contração-Expansão (β)	Variável[2]	Variável[3]
Limites de β	β_{MIN} e β_{MAX} [4]	β_{MIN} e β_{MAX} [5]
Cálculo de *LIP*	Fixo (Equação 2.27)	Fixo e Variável[6]

As observações encontradas na Tabela 3.3 são:

1. Os valores de R_{c1} e R_{c2} utilizam geradores de números aleatórios uniforme entre zero e um, tanto para as partículas originais quanto para as réplicas;
2. O valor de β pode adquirir um comportamento de decrescimento linear ou randômico, similar ao fator de inércia do PSO (vide Equações 2.30 e 2.31);
3. O fator β possui o mesmo comportamento descrito no ponto 2, com a diferença de que as réplicas terão valor de β submetidos ao processo de mutação, gerando assim novos valores de inércia ($m\beta_{(i)}$);
4. Os limites de β dependem da equação de atualização do fator de contração-expansão utilizada. No caso da Equação 2.30, os valores de β_{MIN} e β_{MAX} serão 0 e 0.5;
5. Os valores do fator de contração-expansão das réplicas são inicializados através de um gerador de números aleatórios uniforme em um intervalo de zero a um;
6. A equação de atualização de posição no enxame mestre do QPSO+EE segue o mesmo critério adotado pelo algoritmo PSO+EE, utilizando o procedimento de seleção baseado no fator de migração (Φ). Mais detalhes no subtópico 4.41.

O algoritmo QPSO+EE sempre irá gerar novos valores de fator de contração-expansão para as réplicas tendo por base os valores atuais da respectiva variável. A Equação 3.23 demonstra o processo de mutação modificado para o fator de contração-expansão para as réplicas, gerando o parâmetro $m\beta_{(i)}$ utilizado pelo algoritmo QPSO+EE.

$$m\beta_{(i)} = \beta + (1 + \sigma N(0,1)) \tag{3.23}$$

Junto com o procedimento de mutação, faz-se necessária modificações no processo de atualização dos valores de posição da réplica, onde a constante de probabilidade θ é usada como parâmetro de escolha na seleção do cálculo de *LIP*. Outra mudança está na utilização do melhor global submetido ao processo de pertubação descrito na equação 2.21 no cálculo de *LIP*, proporcionando novos valores de referência do melhor valor global no intuito de aprimorar a busca. O cálculo do *LIP* modificado para as réplicas é descrito pela Equação 3.24, derivada das Equações 2.23 e 2.28.

$$rLIP_i = \begin{cases} {}^{(R_{(c1)}P_i)+(R_{(c2)}mP_{g(i)})}/_{(R_{(c1)}+R_{(c2)})} & \text{se } rand < \theta \\ {}^{(R_{(c1)}P_i)}/_{R_{(c1)}} & \text{se } rand > \theta \end{cases} \tag{3.24}$$

Com a obtenção de $rLIP_i$ (), os valores de posição para as réplicas são geradas através da Equação 3.25, derivada das Equações 2.23 e 2.36.

$$rX_i(t+1) = \begin{cases} rLIP_i + m\beta_{(i)} \left| P_{M(i)} - X_i(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand < 0.5 \\ rLIP_i - m\beta_{(i)} \left| P_{M(i)} - X_i(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand > 0.5 \end{cases} \tag{3.25}$$

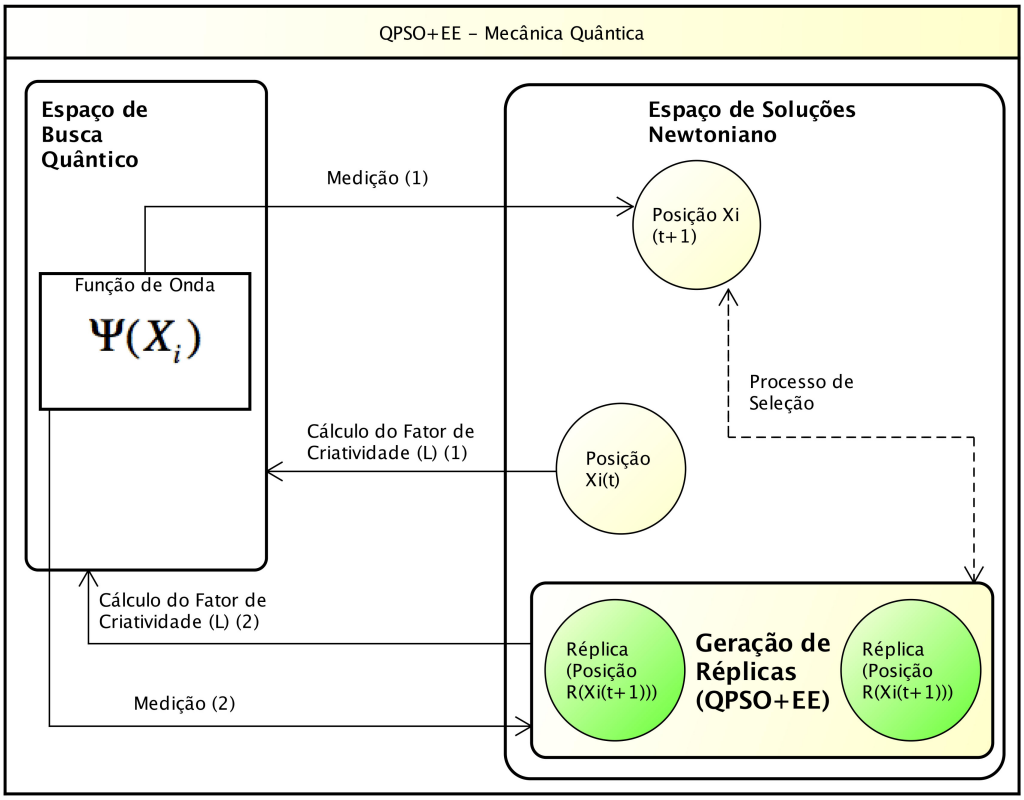
Onde:

- $rX_i(t+1)$: Nova posição da réplica gerada pela partícula i;
- $X_i(t)$: Posição atual da partícula original i;
- $rLIP_i$: Valor de *LIP* associado as réplicas;
- $m\beta_{(i)}$: Fator de contração-expansão submetido ao processo de mutação;
- θ: Constante de probabilidade (referência para topologia de estrela).

Dado que os algoritmos QPSO e QPSO+EE não possuem constantes pré-estabelecidas pelo usuário ou fatores de interação social e individual atuando no processo, as réplicas utilizam apenas o fator de contração-expansão no processo de mutação, bem como operações baseadas em estratégias evolutivas no cálculo do *LIP* para a geração de novos valores de posição. A respeito do procedimento de atualização de posição das partículas originais, utiliza-se a mesma equação do algoritmo QPSO original (vide Equação 2.36).

A Figura 3.7 ilustra a inserção das estratégias evolutivas no espaço de busca do algoritmo QPSO+EE.

Figura 3.7: Ilustração do espaço de busca do algoritmo QPSO+EE



Assim como o algoritmo PSO+EE, no intuito de diminuir as chances do processo de busca sofrer estagnação local pela baixa variabilidade dos valores de β a cada iteração, os resultados obtidos pelo processo de busca são independentes dos valores mutados de β ($m\beta_{(i)}$) pois, caso as réplicas não gerem uma partícular melhor do que a original, a partícula original ainda contará com o valor do fator de contração-expansão original, bem como cálculo de *LIP* original.

Com as adaptações propostas pelo algoritmo QPSO+EE, obtem-se uma solução computacional que agrega as vantagens encontradas tanto no QPSO clássico quanto no modelo de estratégias evolutivas proposto por [Miranda & Fonseca 2002] em uma única solução, onde espera-se um melhor aproveitamento do espaço de busca, bem como uma convergência mais rápida em relação ao algoritmo QPSO original.

3.3.1 QPSO e QPSO+EE Aplicados ao Enxame Mestre (Algoritmos COQMSO e CQEMSO)

Assim como nos enxames escravos, os algoritmos QPSO e QPSO+EE podem ser implementados em um enxame mestre. Utilizando a abordagem dos algoritmos COQMSO e CQEMSO, a solução proposta apresenta modificações significativas, bem como novos mecanismos que auxiliam no processo de otimização do enxame mestre. É importante ressaltar que, dada a estrutura encontrada no *LIP*, bem como a abordagem escolhida para seleção do parâmetro a ser utilizado como limite estabelecido pelo melhor global, este documento apresenta apenas a versão competitiva do processo de otimização no enxame mestre. Os principais recursos usados no QPSO+EE aplicado ao enxame mestre são:

- ***Inclusão dos componentes de melhor global dos enxames escravos ao cálculo de LIP***: Semelhantemente aos componentes de melhor local e global do enxame mestre ($P_{i(M)}$ e $P_{g(M)}$), o melhor resultado encontrado pelos enxames escravos ($P_{g(S)}$) é inserido no cálculo de atualização do *LIP*;
- ***Adição do fator de migração ao cálculo de LIP (abordagem competitiva)***: O processo de busca e otimização no enxame mestre utiliza os valores do fator de migração na seleção do melhor valor global a ser usado no cálculo do *LIP*;
- ***Pertubação do melhor global encontrado pelo enxame mestre (réplicas)***: Os valores de posição de $P_{g(M)}$ são submetidos a um processo de pertubação ($mP_{g(M)}$) com objetivo de gerar novos referenciais para o processo de busca das réplicas. Diferentemente do algoritmo PSO+EE, que submete apenas o melhor global do enxame mestre, o melhor valor encontrado pelos enxames escravos ($mP_{g(S)}$) também estará sujeito ao processo de pertubação, embora não simultâneamente;
- ***Novas equações para atualização de posição (originais e réplicas)***: A adição das estratégias evolutivas no enxame mestre utiliza uma versão modificada da Equação 2.34 e implementada com base nas equações propostas por [Niu et al. 2005] (2.37).

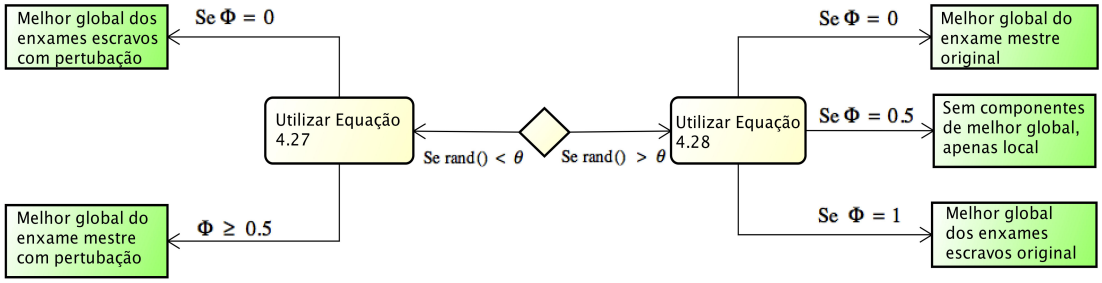
As principais modificações realizadas algoritmo QPSO+EE para a execução em um enxame mestre são expostas nos subtópicos posteriores.

LIP com Pertubação de Melhor Global (Réplicas)

Para a utilização das estratégias evolutivas do QPSO+EE no enxame mestre, o fator de migração (Φ) assume o papel de parâmetro de seleção entre o melhor global do enxame mestre ($mP_{g(M)}$) ou dos enxames escravos ($mP_{g(S)}$). A variável escolhida é então submetida a um processo de pertubação e inserida no cálculo de *LIP*.

Tais alterações tem por objetivo a utilização do melhor valor de referência global para a atualização do *LIP*, bem como a obtenção de novos valores a ser utilizados como parâmetros de melhor global através do processo de pertubação, permitindo assim um melhor aproveitamento do espaço de busca. A Figura 3.7 demonstra o processo de escolha da equação para o cálculo do *LIP*, sendo as operações expostas descritas logo abaixo.

Figura 3.8: Seleção da equação de *LIP* para as réplicas do enxame mestre



A Equação 3.26 expõe o procedimento de pertubação do melhor global encontrado pelos enxames escravos ($P_{g(S)}$). O fator de pertubação ω_i é obtido através da Equação 2.20.

$$mP_{g(S)} = P_{g(S)} + (1 + \omega_i N(0,1)) \tag{3.26}$$

O cálculo de *LIP* é descrito através do seguinte procedimento: Se o valor obtido por um gerador de números aleatórios uniformes entre zero e um (*rand*) for menor do que a constante θ, o cálculo de *LIP* modificado para as réplicas é definido pela Equação 3.27. Caso contrário, os valores de melhor global do enxame mestre e dos enxames escravos não são submetidos ao processo de pertubação e o cálculo de *LIP* modificado para as réplicas passa a ser definido pela Equação 3.28.

$$LIP_{i(M)} = \begin{cases} {}^{(R_{(c1)}P_i)+(R_{(c2)}mP_{g(S)})}/_{(R_{(c1)}+R_{(c2)})} & \text{se } \Phi = 0 \\ {}^{(R_{(c1)}P_i)+(R_{(c2)}mP_{g(M)})}/_{(R_{(c1)}+R_{(c2)})} & \text{se } \Phi \geq 0.5 \end{cases} \tag{3.27}$$

$$LIP_{i(M)} = \begin{cases} {}^{(R_{(c1)}P_i)+(R_{(c2)}P_{g(M)})}/_{(R_{(c1)}+R_{(c2)})} & \text{se } \Phi = 0 \\ {}^{(R_{(c1)}P_i)}/_{R_{(c1)}} & \text{se } \Phi = 0.5 \\ {}^{(R_{(c1)}P_i)+(R_{(c2)}P_{g(S)})}/_{(R_{(c1)}+R_{(c2)})} & \text{se } \Phi = 1 \end{cases} \tag{3.28}$$

Após a obtenção dos valores de *LIP*, calcula-se a posição da réplica através da Equação 3.29.

$$rX_{i(M)}(t+1) = \begin{cases} LIP_{i(M)} + m\beta_{(i)} \left| P_{M(i)} - X_{i(M)}(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand < 0.5 \\ LIP_{i(M)} - m\beta_{(i)} \left| P_{M(i)} - X_{i(M)}(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand > 0.5 \end{cases} \quad (3.29)$$

Onde:

- $LIP_{i(M)}$: Valor de *LIP* do enxame mestre para as réplicas partícula *i*;
- $mP_{g(S)}$: Melhor valor global obtido pelos enxames escravos submetido ao processo de pertubação.

Atualização de Posição para Partículas Originais

A equação de atualização de posição da partícula original no enxame mestre é similar a encontrada no algoritmo QPSO clássico, porém, com alterações na obtenção do valor de *LIP* no intuito de utilizar o melhor valor encontrado por todo o sistema como referencial global para o cálculo do $LIP_{i(M)}$. As principais diferenças entre o método utilizado nas réplicas e na partícula originais são a utilização do fator de migração como único parâmetro de seleção entre os valores de melhor global dos enxames mestre e escravos, bem como a utilização do valor original do coeficiente de contração-expansão (β). A Equação 3.30 demonstra o cálculo do *LIP* para as partículas originais.

$$LIP_{i(M)} = \begin{cases} {(R_{(c1)}P_i)+(R_{(c2)}P_{g(S)})}/{(R_{(c1)}+R_{(c2)})} & \text{se } \Phi = 0 \\ {(R_{(c1)}P_i)+(R_{(c2)}P_{g(M)})}/{(R_{(c1)}+R_{(c2)})} & \text{se } \Phi \geq 0.5 \end{cases} \quad (3.30)$$

Após a obtenção dos valores de *LIP*, calcula-se a posição das partículas originais através da Equação 3.31.

$$X_{i(M)}(t+1) = \begin{cases} LIP_{i(M)} + \beta \left| P_{M(i)} - X_i(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand < 0.5 \\ LIP_{i(M)} - \beta \left| P_{M(i)} - X_i(t) \right| \ln(\frac{1}{R_{(u)}}) & \text{se } rand > 0.5 \end{cases} \quad (3.31)$$

Onde:

- $LIP_{i(M)}$: Valor de *LIP* do enxame (partícula *i*);

3.4 Condições de Contorno de Equivalência e Espelhamento

As operações de reposicionamento de partículas baseadas nos métodos de espelhamento e equivalência de coordenadas surgem como duas novas abordagens de condições de contorno, cujo o objetivo está na manipulação de partículas que escapam do espaço de busca sem que necessariamente sejam reposicionadas nos limites violados. As próximas subseções abodarão cada uma dessas técnicas, bem como a utilização no contexto dos algoritmos abordados neste documento

3.4.1 Espaços Virtuais de Busca

As condições de contorno propostas neste documento utilizam a abordagem de espaços virtuais para obter novos valores de posição. Diferentemente do método tradicional de condição de contorno, onde as coordenadas de posição localizadas fora do espaço de busca representam soluções inviáveis, os espaços virtuais indexam valores válidos a tais locais e utiliza-se de estratégias específicas para manipular posições. As Figuras 3.9 e 3.10 mostram o mapeamento dos espaços virtuais com base nos procedimentos de equivalência e espelhamento.

Figura 3.9: Organização de coordenadas no espaço de busca real e virtual para o procedimento de equivalência

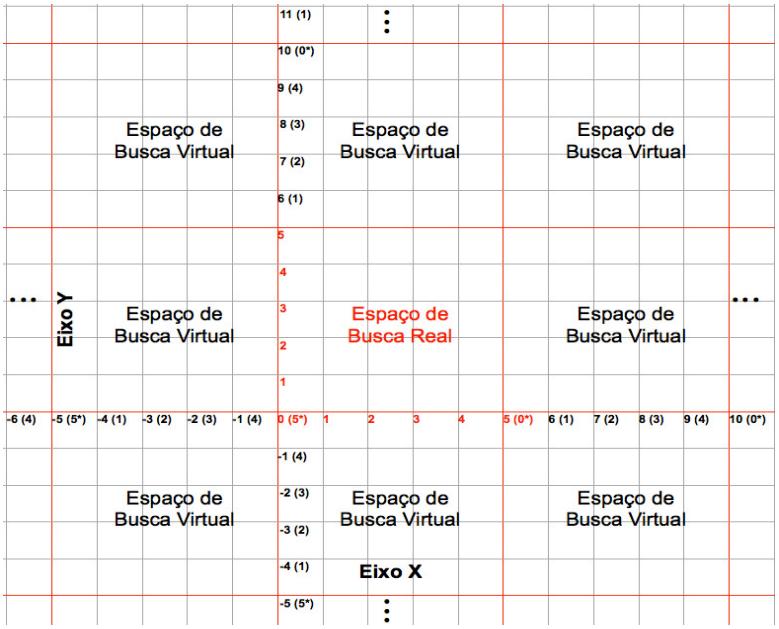
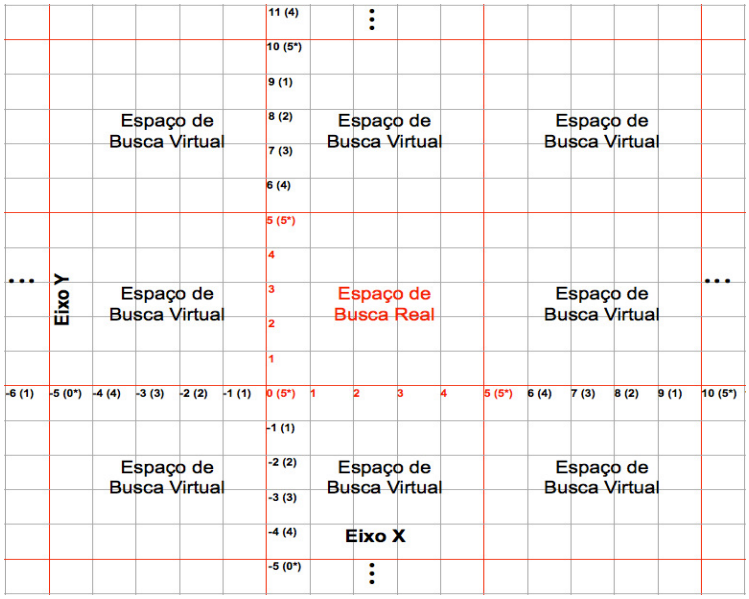


Figura 3.10: Organização de coordenadas no espaço de busca real e virtual para o procedimento de espelhamento



De maneira geral, um espaço de busca virtual consiste na representação do espaço de busca real em locais além dos limites de posição mínimo e máximo, os quais serão manipulados através das operações de espelhamento e equivalência. Tais operações são descritas abaixo e detalhadas nos subtópicos posteriores.

- ***Equivalência***: Caso a partícula ultrapasse seu limite mínimo (X_{MIN}) ou máximo (X_{MAX}), a posição excedida receberá os seguintes valores:

  1. ***Menor do que*** X_{MIN}: A nova posição será igual a X_{MAX} subtraído pela distância excedida em relação a X_{MIN};
  2. ***Maior do que*** X_{MAX}: A nova posição será igual a X_{MIN} somado pela distância excedida em relação a X_{MAX}.

- ***Espelhamento***: Caso a partícula ultrapasse seu limite mínimo (X_{MIN}) ou máximo (X_{MAX}), a posição excedida receberá os seguintes valores:

  1. ***Menor do que*** X_{MIN}: A nova posição será igual a X_{MIN} somado pela distância excedida em relação a X_{MIN};
  2. ***Maior do que*** X_{MAX}: A nova posição será igual a X_{MAX} subtraído pela distância excedida em relação a X_{MAX}.

3.4.2 Equivalência de Posicionamento

O procedimento de equivalência de posições consiste na reinserção da partícula no espaço de busca, atribuindo novas coordenadas geradas a partir do valor de posição equivalente dentro do espaço de busca virtual a qual se encontra. Em termos de movimento, ao atravessar o limite estabelecido, a partícula "retorna" ao espaço de busca real pelo limite inverso e sua nova posição é definida pela distância excedida em relação ao limite violado. O procedimento de equivalência aplicado aos valores de posição das partículas é descrito pelas equações abaixo. As Equações 3.32 e 3.33 mostram o processo de equivalência de coordenadas mínimo e máximo respectivamente[1].

$$Eq_{MIN(i)} = X_{MAX} - ((X_{MAX} - X_i(t+1)) - floor\left(\frac{(X_{MAX} - X_i(t+1))}{(X_{MAX} - X_{MIN})}\right)(X_{MAX} - X_{MIN})) \tag{3.32}$$

$$Eq_{MAX(i)} = X_{MIN} + ((X_i(t+1) - X_{MIN}) - floor\left(\frac{(X_i(t+1) - X_{MIN})}{(X_{MAX} - X_{MIN})}\right)(X_{MAX} - X_{MIN})) \tag{3.33}$$

A Equação 3.34 mostra o processo de condições de contorno para equivalência de coordenadas.

$$X_i(t+1) = \begin{cases} Eq_{MIN(i)} & \text{se } X_i(t+1) < X_{MIN} \\ X_i(t+1) & \text{se } X_{MIN} \leq X_i(t+1) \leq X_{MAX} \\ Eq_{MAX(i)} & \text{se } X_i(t+1) > X_{MAX} \end{cases} \tag{3.34}$$

Com o reposicionamento das partículas, o valor de velocidade sofre uma diminuição, tendo seu valor multiplicado pela constante δ. As vantagens da desaceralção da partícula são a diminuição da probabilidade desta de escapar do espaço de busca real, além de manter a direção da partícula para os resultados de referência (melhor local e melhor global). A Equação 3.35 demonstra as condicões de contorno para os valores de velocidade.

$$V_i(t+1) = \begin{cases} V_i(t+1)\delta R & \text{se } X_i(t+1) < X_{MIN} \\ V_i(t+1) & \text{se } X_{MIN} \leq X_i(t+1) \leq X_{MAX} \\ V_i(t+1)\delta R & \text{se } X_i(t+1) > X_{MAX} \end{cases} \tag{3.35}$$

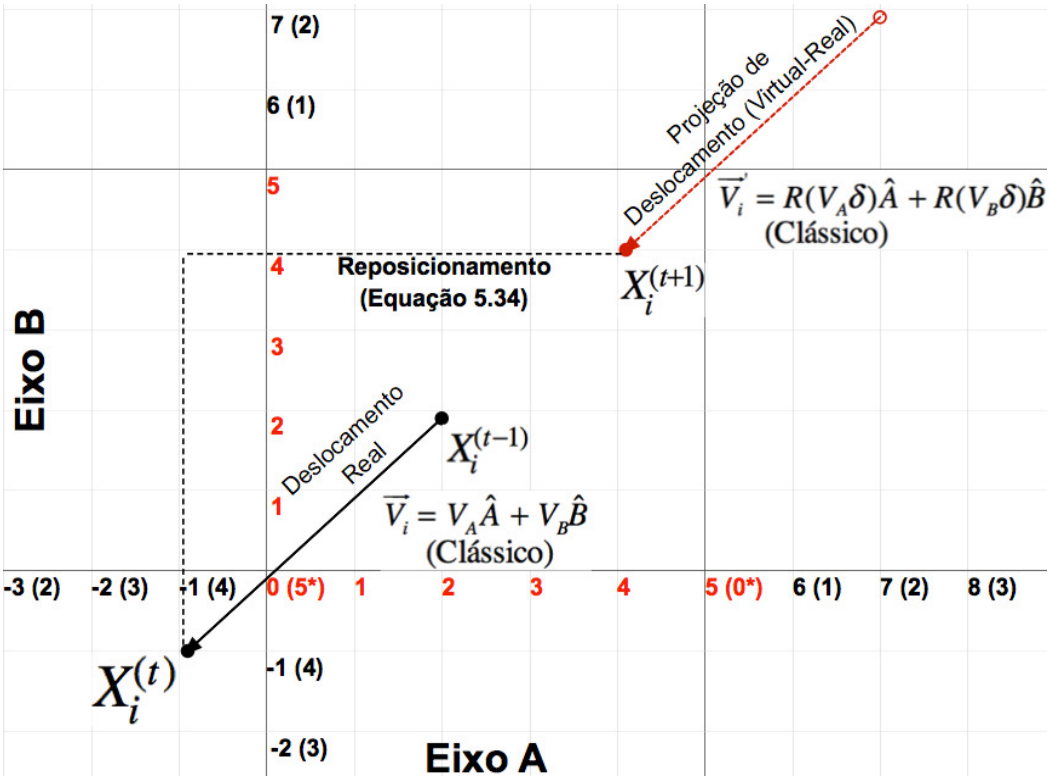
[1] A função *floor()* encontrada nas Equações 3.32 e 3.33 elimina a parte decimal do valor resultante da operação especificada.

Onde:

- $Eq_{MIN(i)}$ *e* $Eq_{MAX(i)}$: Valores de posição submetidos ao processo de equivalência de coordenadas mínimo ou máximos da partícula i;
- $V_i(t+1)$: Novo valor de velocidade para a partícula i;
- δ: Fator de diminuição de velocidade;
- R: Número aleatório gerado uniformemente entre o limite mínimo definido pelo usuário até 1.

Com o procedimento de equivalência, é possível obter um melhor aproveitamento do espaço de busca, dado que uma partícula fora dos limites estabelecidos irá possuir coordenadas válidas através do mapeamento do espaço de busca virtual. A Figura 3.11 mostra o processo de equivalência em um espaço bidimensional, onde a partícula excede ambas as dimensões (X e Y). É importante ressaltar que o termo ”clássico” encontrado junto as equações referentes ao vetor velocidade indicam que tal operação é realizada somente pelos algoritmos baseados na mecânica newtoniana (PSO e EPSO).

Figura 3.11: Ilustração do procedimento de equivalência de posições



3.4.3 Espelhamento de Posicionamento

O procedimento de espelhamento ou reflexão [2] de posições consiste na reinserção da partícula no espaço de busca, atribuindo novas coordenadas geradas a partir do valor de posição inverso dentro do espaço de busca virtual a qual se encontra. Sob uma perspectiva física, caso haja violação de limites, o comportamento da partícula assemelha-se a de um objeto colidindo em uma parede e retornando para o espaço de busca real, onde a posição resultante será o valor da distância excedida em relação ao limite violado. As Equações 3.36 e 3.37 mostram o processo de espelhamento de coordenadas mínimo e máximo respectivamente[3].

$$Esp_{MIN(i)} = X_{MIN} + ((X_{MIN} - X_{i(M)}(t+1)) - floor\left(\frac{(X_{MIN} - X_{i(M)}(t+1))}{(X_{MAX} - X_{MIN})}\right)(X_{MAX} - X_{MIN})) \tag{3.36}$$

$$Esp_{MAX(i)} = X_{MAX} - ((X_i(t+1) - X_{MAX}) - floor\left(\frac{(X_i(t+1) - X_{MAX})}{(X_{MAX} - X_{MIN})}\right)(X_{MAX} - X_{MIN})) \tag{3.37}$$

A Equação 3.38 mostra o processo de condições de contorno para espelhamento de coordenadas.

$$X_i(t+1) = \begin{cases} Esp_{MIN(i)} & \text{se } X_i(t+1) < X_{MIN} \\ X_i(t+1) & \text{se } X_{MIN} \leq X_i(t+1) \leq X_{MAX} \\ Esp_{MAX(i)} & \text{se } X_i(t+1) > X_{MAX} \end{cases} \tag{3.38}$$

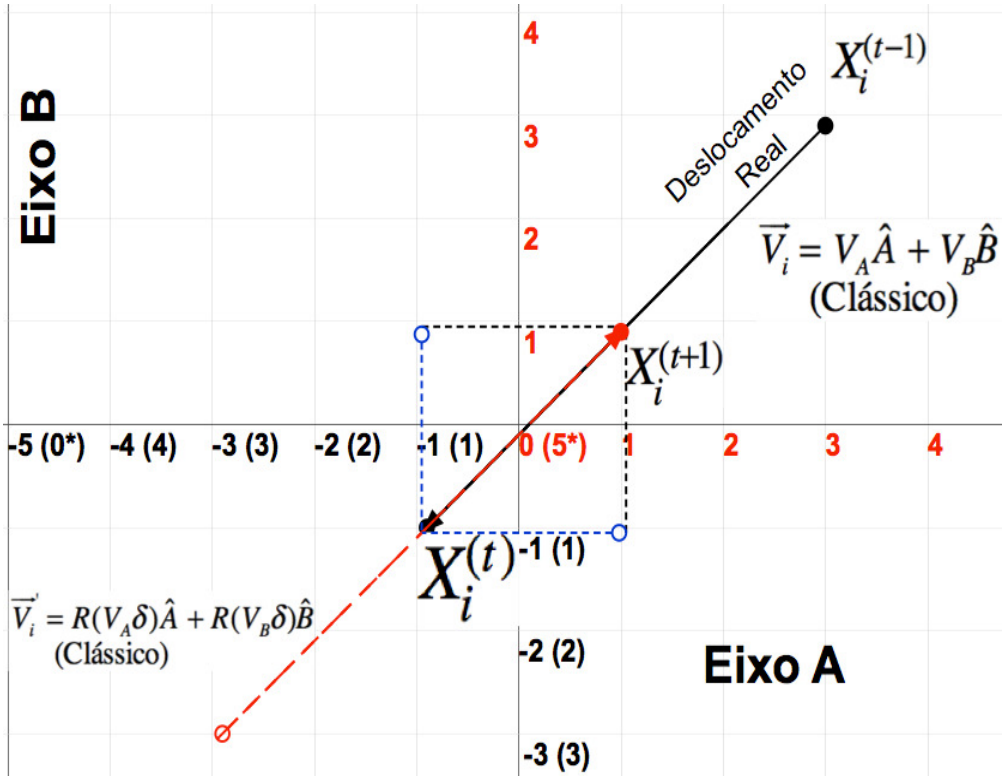
[2] O procedimento de reflexão abordado neste documento difere das condições de contorno de limite descritas por [Xu & Rahmat-Samii 2007], onde a reflexão ocorre apenas no vetor de velocidade, sendo que o valor de posição passa a ser o valor do limite violado

[3] A função *floor()* encontrada nas Equações 3.36 e 3.37 elimina a parte decimal do valor resultante da operação especificada.

Semelhantemente ao processo de equivalência, após o reposicionamento das partículas, o valor de velocidade sofre uma diminuição, tendo seu valor multiplicado pela constante δ (vide Equação 3.35). A desaceleração da partícula é resultante do ”impacto” sofrido ao alcançar o limite e tem como objetivo diminuir a probabilidade desta de escapar do espaço de busca real, otimizar as buscas em coordeandas próximas ao limite no decorrer das iterações[4], além de manter a direção da partícula para os resultados de referência (melhor local e melhor global).

A Figura 3.12 mostra o processo de espelhamento em um espaço bidimensional, onde a partícula excede ambas as dimensões (X e Y). É importante ressaltar que o termo ”clássico” encontrado junto as equações referentes ao vetor velocidade indicam que tal operação é realizada somente pelos algoritmos de inteligência de enxame baseados na mecânica newtoniana (PSO e EPSO).

Figura 3.12: Ilustração do procedimento de espelhamento de posições



[4]Esta vantagem pode ser melhor aproveitada caso o fator de inércia assuma um comportamente decrescente, como exposto na Equação 2.1

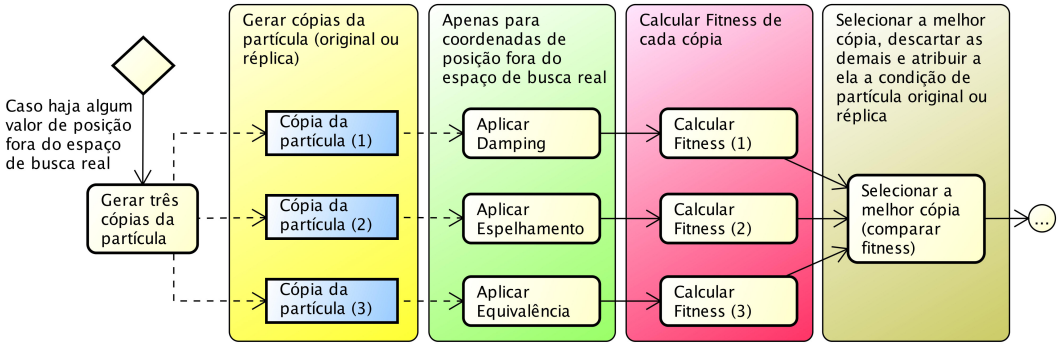
3.4.4 Utilização nos Algoritmos PSO+EE e QPSO+EE

Para a implementação das condições de contorno de equivalência e espelhamento de coordenadas nos algoritmos expostos neste documento, cada partícula (incluindo as réplicas) terá três valores de *fitness* distintos, onde cada um representa uma solução obtida com o uso de condições de contorno diferentes. Ao final do cálculo de velocidade e posição, a melhor das três soluções é escolhida para a operação de obtenção do melhor local.

- ***Condições de contorno de fronteira Damping (Fitness 1)***: Caso haja violações de limite, a primeira solução da partícula utilizará o método de correção "*Damping*". Em relação a posições fora do espaço de busca real, este resultado enfatiza soluções com coordenadas localizadas nos limites mínimo ou máximo;
- ***Condições de contorno baseado em equivalência de posições (Fitness 2)***: Caso haja violações de limite, a segunda solução da partícula utilizará o reposicionamento por equivalência. Em relação as posições fora do espaço de busca real, este resultado enfatiza soluções com coordenadas convertidas do espaço de busca virtual;
- ***Condições de contorno baseado em espelhamento ou reflexão de posições (Fitness 3)***: Caso haja violações de limite, a terceira solução da partícula utilizará o reposicionamento por espelhamento. Em relação as posições fora do espaço de busca real, este resultado enfatiza soluções com coordenadas resultantes do processo de reação ao "impacto" da partícula com o limite violado, bem como valores de posição próximos aos limites no decorrer das iterações;

A Figura 3.13 demonstra os procedimentos de escolha das condições de contorno aplicadas nos algoritmos PSO, PSO+EE, EPSO e QPSO, bem como em suas versões multi-enxame.

Figura 3.13: Geração de cópias, aplicação de condições de contorno e seleção de melhor resultado



O algoritmo 4 demonstra a geração das cópias da partícula original com base no procedimento exposto na Figura 3.13.

Algoritmo 4: Aplicação de condições de contorno (original)

if $X_i(t+1)$ *violar limite mínimo ou máximo do espaço de busca* **then**
- Aplicar condições de contorno "*Damping*" aos valores de posição para a cópia 1 da partícula original (Equação 2.11);
- Aplicar condições de contorno "*Damping*" aos valores de velocidade para a a cópia 1 partícula original (Equação 2.10);
- Aplicar condições de contorno de equivalência aos valores de posição para a cópia 2 da partícula original (Equação 3.34);
- Aplicar condições de contorno de equivalência aos valores de velocidade para a a cópia 2 da partícula original (Equação 3.35);
- Aplicar condições de contorno de espelhamento aos valores de posição para a cópia 3 da partícula original (Equação 3.38);
- Aplicar condições de contorno de espelhamento aos valores de velocidade para a a cópia 3 da partícula original (Equação 3.35);

O algoritmo 5 demonstra a geração das cópias das réplicas com base no procedimento exposto na Figura 3.13.

Algoritmo 5: Aplicação de condições de contorno (réplicas)

if $X_i(t+1)$ *violar limite mínimo ou máximo do espaço de busca* **then**
- Aplicar condições de contorno "*Damping*" aos valores de posição para a cópia 1 da réplicas (Equação 2.11);
- Aplicar condições de contorno "*Damping*" aos valores de velocidade para a a cópia 1 réplicas (Equação 2.10);
- Aplicar condições de contorno de equivalência aos valores de posição para a cópia 2 da réplicas (Equação 3.34);
- Aplicar condições de contorno de equivalência aos valores de velocidade para a a cópia 2 da réplicas (Equação 3.35);
- Aplicar condições de contorno de espelhamento aos valores de posição para a cópia 3 da réplicas (Equação 3.38);
- Aplicar condições de contorno de espelhamento aos valores de velocidade para a a cópia 3 da réplicas (Equação 3.35);

Com a utilização de três soluções obtidas com condições de contorno distintas, obtem-se um algoritmo de busca mais eficiente, onde é possível explorar soluções encontradas nos limites, em espaços virtuais convertidos em espaços reais e coordenadas próximas aos limites de cada dimensão.

3.5 CEMSO e CQEMSO Sob a Arquitetura CUDA

Com a utilização de uma abordagem multi-enxame, faz-se necessário a utlização de um ambiente paralelo e/ou distribuído para a obtenção de uma solução com tempo de processamento viável, onde várias instâncias de enxames são executadas concorrentemente. A implementação dos algoritmos propostos, baseados em uma versão modificada do modelo de [Niu et al. 2005] e projetada sob a arquitetura CUDA possibilita uma melhor representação do processo de movimentação das partículas através da seguinte relação:

- ***Uma thread, uma partícula***: Cada *thread* é responsável pela manipulação de uma partícula (cardinalidade 1:1);
- ***Um block, um enxame***: Cada *block* fica responsável por um enxame escravo e/ou mestre em *grids* distintos (cardinalidade 1:1).

Dada a cardinalidade acima, faz-se necessário a especificação das operações que serão paralelizadas ou mantidas em processamento serial. A Tabela 3.4 mostra o tipo de execução[5] das operações referentes ao processo de busca e otimização dos algoritmos COMSO, COEMSO, COQMSO, CEMSO e CQEMSO sob a arquitetura CUDA. As células que contém o termo $P(T,B)$ indicam que o procedimento especificado é executado em paralelo, tanto a nível de partícula (*thread*) quanto a nível de enxame (*block*), enquanto o termo $S(CPU)$ e $S(GPU)$ indicam processamento serial na CPU ou na GPU.

Tabela 3.4: Execução de operações relacionadas aos algoritmos COMSO, COEMSO, COQMSO, CEMSO e CQEMSO sob a arquitetura CUDA

Algoritmo	COMSO	COEMSO	COQMSO	CEMSO	CQEMSO
Obtenção de Média Local[1]	—	—	$S(CPU)$	—	$S(CPU)$
Cálculo de Fator de Inércia	$S(CPU)$	$S(CPU)$	—	$S(CPU)$	—
Cálculo de Fator β	—	—	$S(CPU)$	—	$S(CPU)$
Atualização de Velocidade (Original)	$P(T,B)$	$P(T,B)$	—	$P(T,B)$	—
Atualização de Posição (Original)	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$
Condições de Contorno (Original)	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$
Atualização de Velocidade (Réplica)	$P(T,B)$	$P(T,B)$	—	$P(T,B)$	—
Atualização de Posição (Réplica)	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$
Condições de Contorno (Réplica)	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$
Cálculo de *LIP*	—	—	$P(T,B)$	—	$P(T,B)$
Cálculo de *Fitness*	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$
Obtenção do Melhor Local	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$	$P(T,B)$
Obtenção do Melhor Global	$S(GPU)$	$S(GPU)$	$S(GPU)$	$S(GPU)$	$S(GPU)$

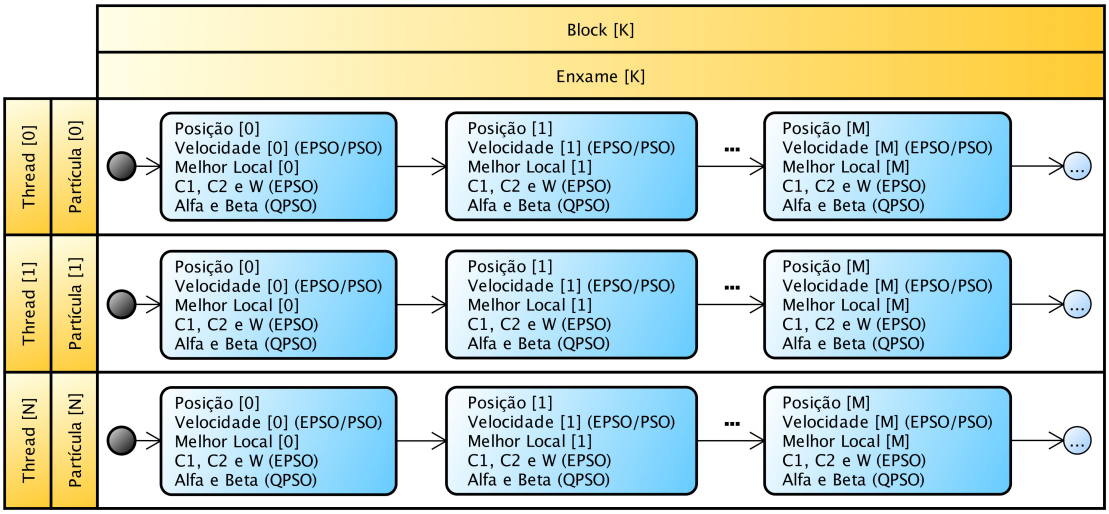
[5] Mais detalhes podem ser encontrados a partir da seção 5.7 e nos Apêndices

[1] Média dos enxames escravos e mestre

Organização de *Threads* e Blocks

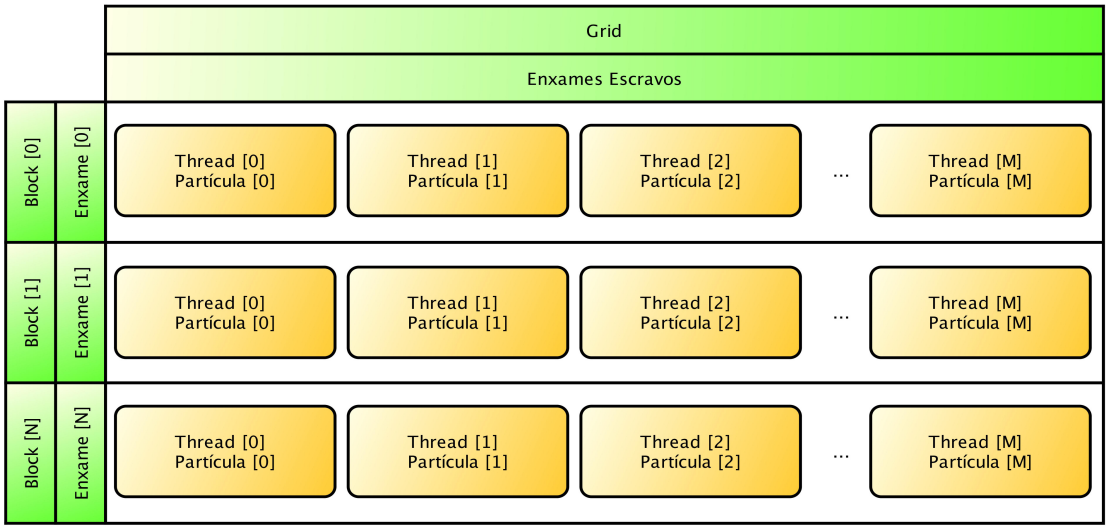
A Figura 3.14 mostra a organização das partículas nos enxames escravos em relação as *threads* nos algoritmos CEMSO, CQEMSO, COQMSO, COMSO e COEMSO sob a arquitetura CUDA.

Figura 3.14: Organização das partículas no *block* em CUDA (enxames escravos)



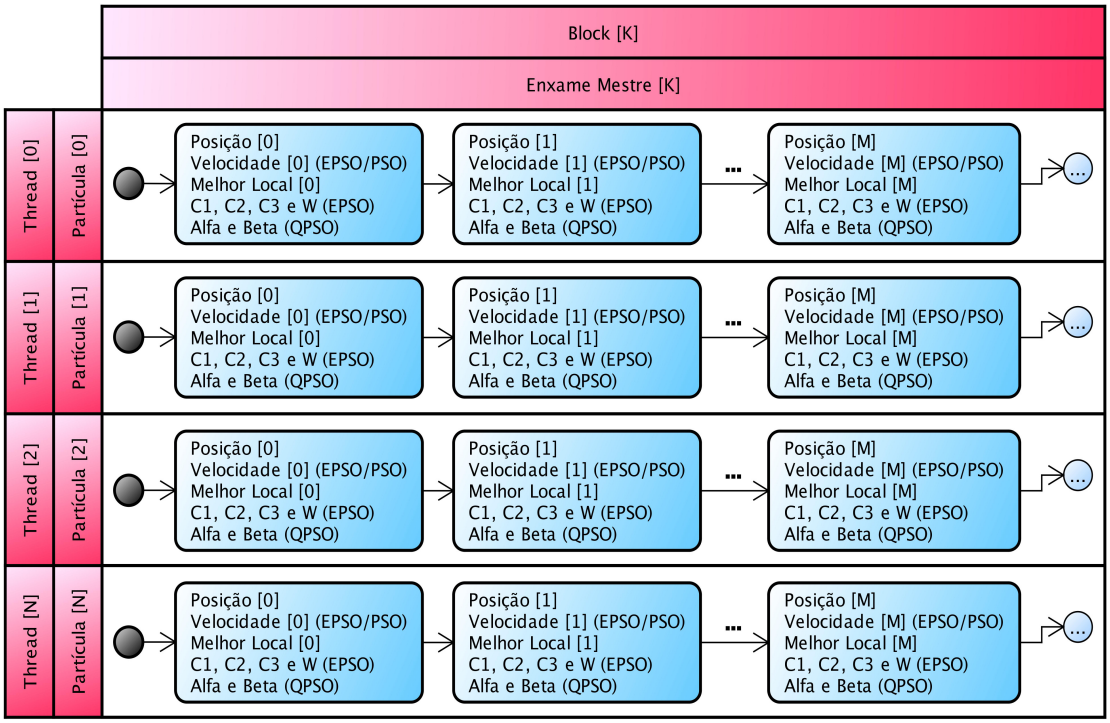
A Figura 3.15 mostra a organização dos enxames escravos em relação aos *blocks* e ao *grid* nos algoritmos CEMSO, CQEMSO, COQMSO, COMSO e COEMSO sob a arquitetura CUDA.

Figura 3.15: Organização dos enxames escravos no *grid* em CUDA



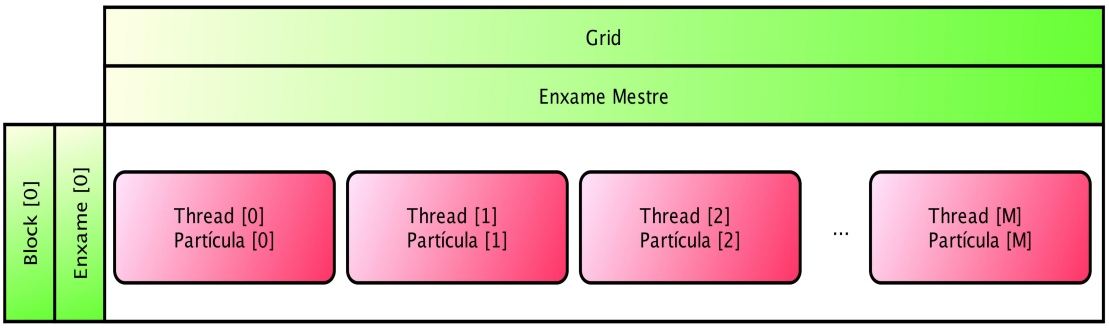
A Figura 3.16 mostra a organização das partículas no enxame mestre em relação as *threads* nos algoritmos CEMSO, CQEMSO, COQMSO, COMSO e COEMSO sob a arquitetura CUDA.

Figura 3.16: Organização das partículas no *block* em CUDA (enxame mestre)



A Figura 3.17 mostra a organização do enxame mestre em relação aos *blocks* e ao *grid* nos algoritmos CEMSO, CQEMSO, COQMSO, COMSO e COEMSO sob a arquitetura CUDA.

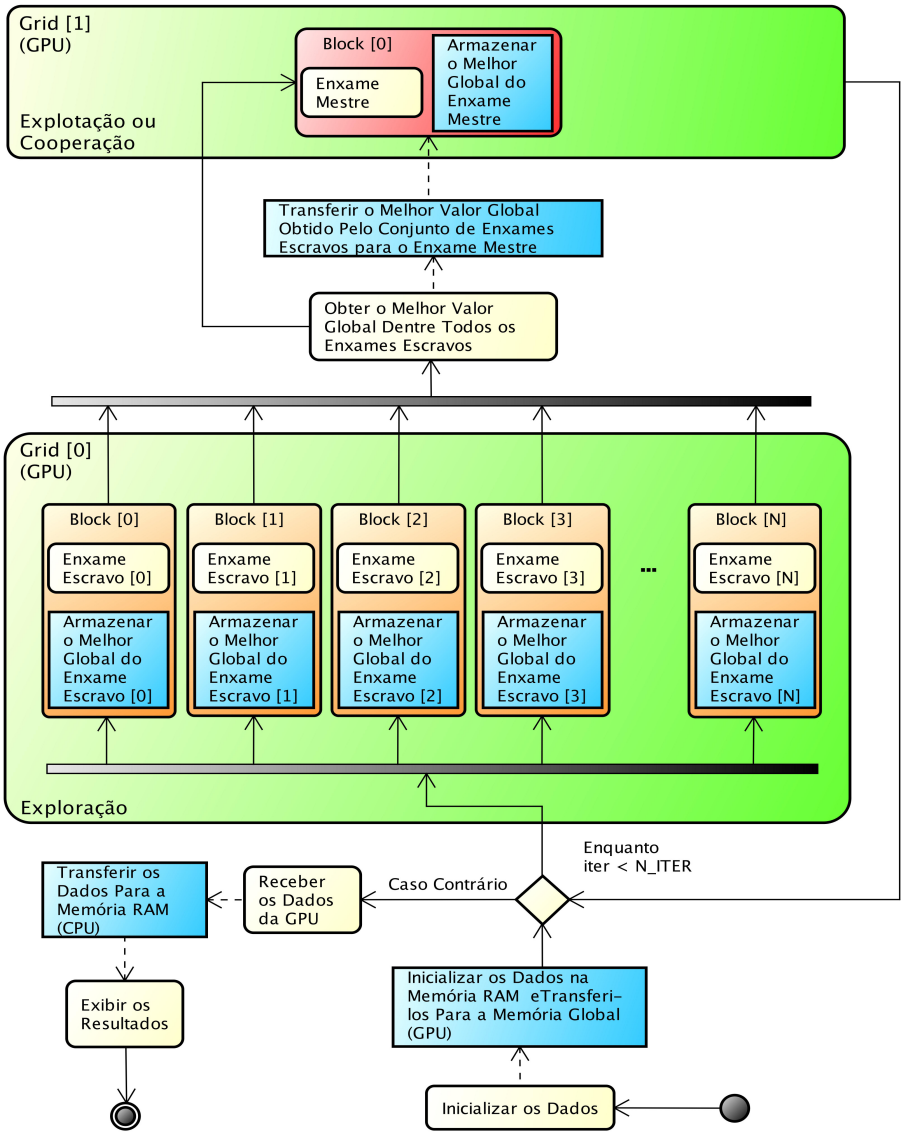
Figura 3.17: Organização do enxame mestre no *grid* em CUDA



Fluxo de Execução do Algoritmo em CUDA

Com os recursos de organização de *threads* através dos *blocks*, é possível obter um cenário de paralelismo que engloba tanto partículas quanto enxames, ou seja, ambas as estruturas são executadas de maneira concorrente. A Figura 3.18 demonstra o ambiente multi-enxame de topologia mestre-escravos modificado para a arquitetura CUDA, desde a inicialização das estruturas de dados na memória para armazenamento dos enxames, até a execução do algoritmo na GPU.

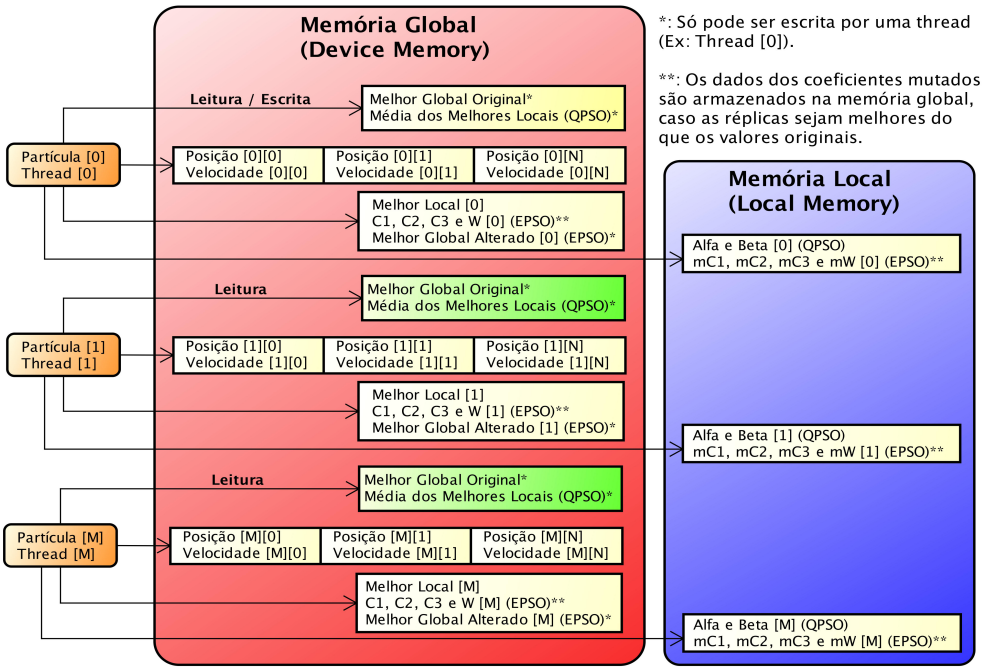
Figura 3.18: Esquema ilustrativo do modelo mestre-escravo de [Niu et al. 2007] utilizado nos algoritmos CEMSO, CQEMSO, COMSO, COEMSO e COQMSO modificado para a arquitetura CUDA



Memórias

O armazenamento dos dados entre os enxames escravos e mestre é realizado através da memória global (*global memory*) com a utilização da memória local para variáveis de instância que necessitam de acesso e armazenamento rápido (e.g. fator de inércia original (w ou $w_{(i)}$), parâmetros de iteração individual e social (mestre e escravos) no algoritmo EPSO ($mC1_{(i)}$, $mC2_{(i)}$ e $mC3_{(i)}$), coeficiente de contração-expansão e fator α para o algoritmo QPSO). O procedimento inicial consiste no carregamento dos dados na memória principal (RAM), sendo posteriormente transferidos para a memória global da GPU, onde serão manipuladas pelas operações de busca e otimização do PSO e variantes. A Figura 3.19 mostra a distribuição de dados dos enxames mestre e escravos em relação as memórias disponíveis na arquitetura CUDA.

Figura 3.19: Distribuição dos dados do enxame (mestre e escravos) às memórias da GPU



Com a organização dos dados e do controle de execução das *threads* na GPU, o estágio seguinte consiste na estruturação dos métodos de busca e otimização abordados neste documento. As próximas seções[6] abordam com detalhes a implementação (pseudocódigo) dos algoritmo multi-enxame de topologia mestre-escravos CEMSO, CQEMSO, bem como de suas bases (PSO+EE, QPSO+EE).

[6] A implementação (pseudocódigo) dos algoritmos COMSO, COEMSO e COQMSO são pode ser encontrada no Apêndice

3.6 Pseudocódigo (Algoritmo PSO+EE)

O algoritmo 6 monstra os procedimentos do PSO+EE com fator de inércia e condições de contorno com base no procedimento exposto na Figura 3.13.

Algoritmo 6: PSO+EE com condições de contorno modificada

```
Alocar memória para estruturas do enxame (posição (X), velocidade (V));
Alocar memória para melhores valores do enxame (local (P_i), global (P_g));
for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
    for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
        Inicializar os valores de velocidade;
        Inicializar os valores de posição;
    Inicializar os valores de fator de inércia;
    Inicializar os valores de interação individual;
    Inicializar os valores de interação social;
    Inicializar os valores para o fator de pertubação do melhor global;
    Avaliar partícula (Fitness);
    Inicializar os melhor local da partícula (P_i);
    Obter o melhor global do enxame (P_g);
for iter ← 1 to (QUANTIDADE DE ITERAÇÕES−1 do
    Atualizar fator de inércia (Equação 2.1 ou 2.2);
    for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
        for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
            Atualizar valor de velocidade da variável j para a partícula original (Equação 2.5);
            Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
            Atualizar valor de posição da variável j para a partícula original (Equação 2.6);
            Aplicar condições de contorno descrita no Algoritmo 4;
        if Houver cópias da partícula com resultados de condições de contorno distintas then
            Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
        Avaliar partícula original (Fitness);
        for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
            Gerar novo valor para o fator de inércia através da mutação (Equação 3.1);
            Gerar novo valor para o fator de interação individual através da mutação (Equação 3.2);
            Gerar novo valor para o fator de interação social através da mutação (Equação 3.3);
            Gerar novo valor para o fator de pertubação através da mutação (Equação 2.20);
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a réplica k (Equação 2.23);
                Aplicar ajuste de velocidade para a réplica k (Equação 2.9);
                Atualizar valor de posição da variável j para a réplica k (Equação 2.25);
                Aplicar condições de contorno descrita no Algoritmo 5;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar réplica (Fitness);
            Comparar réplica e partícula original;
            if fitness(Replica) melhor do que fitness(Original) then
                Substituir partícula original pela réplica (Original = Replica);
                Substituir fitness original por fitness réplica;
        Obter os melhor local da partícula (P_i);
        Obter o melhor global do enxame (P_g);
```

3.7 Pseudocódigo (Algoritmo CEMSO)

O algoritmo 7 mostra a inicialização dos dados nos enxames escravos do algoritmo CEMSO sob a arquitetura CUDA.

Algoritmo 7: CEMSO - Inicialização dos enxames escravos - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de velocidade;
                Inicializar os valores de posição;
            Inicializar os valores de fator de inércia (w_(i)) para os enxames escravos;
            Inicializar os valores de interação individual (C_1(i)) para os enxames escravos;
            Inicializar os valores de interação social (C_2(i)) para os enxames escravos;
            Inicializar os valores para o fator de pertubação do melhor global (ω_i) para os enxames
            escravos;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 8 mostra a inicialização dos dados no enxame mestre do algoritmo CEMSO sob a arquitetura CUDA.

Algoritmo 8: CEMSO - Inicialização do enxame mestre - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de velocidade;
                Inicializar os valores de posição;
            Inicializar os valores de fator de inércia (w_(i)) para o enxame mestre;
            Inicializar os valores de interação individual (C_1(i)) para o enxame mestre;
            Inicializar os valores de interação social (C_2(i)) para o enxame mestre;
            Inicializar os valores de interação social dos enxames escravos (C_3(i)) para o enxame mestre;
            Inicializar os valores para o fator de pertubação do melhor global (ω_i) e atribuir ao enxame
            mestre;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i(M));
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g(M));
        end
    end
end
```

O algoritmo 9 mostra o procedimento de busca e otimização envolvendo as partículas originais e o uso de estratégias evolutivas para a geração das réplicas nos enxames escravos do algoritmo CEMSO sob a arquitetura CUDA.

Algoritmo 9: CEMSO com fator de inércia (enxames escravos) - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a partícula original (Equação 2.5);
                Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
                Atualizar valor de posição da variável j para a partícula original (Equação 2.6 ou 2.24);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
                Gerar novo valor para o fator de inércia atravé da mutação (Equação 3.1));
                Gerar novo valor para o fator de interação individual através da mutação (Equação 3.2));
                Gerar novo valor para o fator de interação social através da mutação (Equação 3.3));
                Gerar novo valor para o fator de pertubação atravé da mutação (Equação 2.20));
                for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                    Atualizar valor de velocidade da variável j para a réplica k (Equação 2.23);
                    Aplicar ajuste de velocidade para a réplica k (Equação 2.9);
                    Atualizar valor de posição da variável j para a réplica k (Equação 2.25);
                    Aplicar condições de contorno descrita no Algoritmo 5;
                if Houver cópias da partícula com resultados de condições de contorno distintas
                then
                    Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
                Avaliar réplica (Fitness);
                Comparar réplica e partícula original;
                if fitness(Replica) melhor do que fitness(Original) then
                    Substituir partícula original pela réplica (Original = Replica);
                    Substituir fitness original por fitness réplica;
            Obter os melhor local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 10 mostra o procedimento de busca e otimização envolvendo as partículas originais e o uso de estratégias evolutivas para a geração das réplicas no enxame mestre do algoritmo CEMSO sob a arquitetura CUDA.

Algoritmo 10: CEMSO com fator de inércia (enxame mestre) - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a partícula original (Equação 2.37 ou 2.39);
                Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
                Atualizar valor de posição da variável j para a partícula original (Equação 2.6);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
                Gerar novo valor para o fator de inércia atravé da mutação (3.1));
                Gerar novo valor para o fator de interação individual através da mutação (Equação 3.2));
                Gerar novo valor para o fator de interação social através da mutação (Equação 3.3));
                Gerar novo valor para o fator de interação social dos enxames escravos através da mutação (Equação 3.5));
                Gerar novo valor para o fator de pertubação atravé da mutação (Equação 2.20));
                for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                    Atualizar valor de velocidade da variável j para a réplica k (Figura 3.6);
                    Aplicar ajuste de velocidade para a réplica k (Equação 2.9);
                    Atualizar valor de posição da variável j para a réplica k (Equação 2.25);
                    Aplicar condições de contorno descrita no Algoritmo 5;
                if Houver cópias da partícula com resultados de condições de contorno distintas then
                    Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
                Avaliar réplica (Fitness);
                Comparar réplica e partícula original;
                if fitness(Replica) melhor do que fitness(Original) then
                    Substituir partícula original pela réplica (Original = Replica);
                    Substituir fitness original por fitness réplica;
            Obter os melhor local da partícula (P_i(M));
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g(M));
                if P_g(S) for melhor que P_g(M) then
                    Atribuir o valor de P_g(S) a P_g(M);
        end
    end
end
```

O Algoritmo 11 mostra a execução de CEMSO na íntegra sob a arquitetura CUDA.

Algoritmo 11: Algoritmo CEMSO em CUDA

Alocar memória para as estruturas dos enxames escravos (posição (X), velocidade (V)) (GPU);
Alocar memória para o melhores valores dos enxames escravos (local (P_i), global (P_g)) (GPU);
Alocar memória para o melhor resultado obtido pelos enxames escravos ($P_{g(S)}$) (CPU e GPU);
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização dos enxames escravos (Algoritmo 7);
Sincronizar GPU;
Inicializar o melhor resultado encontrado pelos enxames escravos ($P_{g(S)}$);
Alocar memória para estruturas do enxame mestre (posição (X), velocidade (V)) (GPU);
Alocar memória para melhores valores dos enxame mestre (local (P_i), global ($P_{g(M)}$)) (GPU);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização do enxame mestre (Algoritmo 8);
Sincronizar GPU;
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
for $iter \leftarrow 1$ **to** *(QUANTIDADE DE ITERAÇÕES*-1 **do**
    Atualizar fator de inércia (Equação 2.1 ou 2.2);
    Executar o *kernel* de busca e otimização enxames escravos (Algoritmo 9);
    Sincronizar GPU;
    Obter o melhor resultado encontrado pelos enxames escravos ($P_{g(S)}$);
    Executar o *kernel* de busca e otimização enxame mestre (Algoritmo 10);
    Sincronizar GPU;
Copiar os valores de melhor global dos enxames escravos para a memória principal (GPU para CPU);
Copiar os valores de melhor global do enxame mestre para a memória principal (GPU para CPU);
Liberar memória;

3.8 Pseudocódigo (Algoritmo QPSO+EE)

O algoritmo 12 mostra os procedimentos do QPSO+EE com fator α, aplicação das estratégias evolutivas para geração de réplicas baseada no algoritmo EPSO e condições de contorno com base no procedimento exposto na Figura 3.13.

Algoritmo 12: QPSO+EE com fator α e condições de contorno modificada

```
Alocar memória para estruturas do enxame (posição (X));
Alocar memória para melhores valores do enxame (local (P_i), global (P_g));
for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
    for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
        Inicializar os valores de posição;
    Inicializar os valores do fator alfa (α);
    Inicializar os valores de fator beta (β);
    Inicializar os valores para o fator de pertubação do melhor global (ω_i);
    Avaliar partícula (Fitness);
    Inicializar os melhor local da partícula (P_i);
    Obter o melhor global do enxame (P_g);
for iter ← 1 to (QUANTIDADE DE ITERAÇÕES−1 do
    Atualizar fator β (Equação 2.32 ou 2.33);
    Atualizar fator α (Equação 2.29 ou 2.30);
    Obter a média dos melhores locais do enxame (P_M(i)) (Equação 2.31);
    for i ← 0 to (QUANTIDADE DE PARTÍCULAS−1) do
        for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
            Atualizar valor de ponto de inclinação (LIP ou p) (Equação 2.28);
            Atualizar valor de posição da variável j (Equação 2.36);
            Aplicar condições de contorno descrita no Algoritmo 4;
        if Houver cópias da partícula com resultados de condições de contorno distintas then
            Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
        Avaliar partícula (Fitness);
        for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
            Gerar novo valor para o fator de contração-expansão (β) através da mutação (Equação
            3.23));
            Gerar novo valor para o fator de pertubação (ω_i) através da mutação (Equação 2.20));
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de ponto de inclinação (LIP ou p) para a réplica k (Figura 3.8);
                Atualizar valor de posição da variável j para a réplica k (Equação 3.25);
                Aplicar condições de contorno descrita no Algoritmo 5;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar réplica (Fitness);
            Comparar réplica e partícula original;
            if fitness(Replica) melhor do que fitness(Original) then
                Substituir partícula original pela réplica (Original = Replica);
                Substituir fitness original por fitness réplica;
        Obter os melhor local da partícula (P_i);
        Obter o melhor global do enxame (P_g);
```

3.9 Pseudocódigo (Algoritmo CQEMSO)

O algoritmo 13 mostra a inicialização dos dados nos enxames escravos do algoritmo CQEMSO sob a arquitetura CUDA.

Algoritmo 13: CQEMSO - Inicialização dos enxames escravos - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de posição;
            Inicializar os valores de contração-expansão (β) para os enxames escravos;
            Inicializar os valores do o fator de pertubação do melhor global (ω_i) para os
            enxames escravos;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 14 mostra a inicialização dos dados no enxame mestre do algoritmo CQEMSO sob a arquitetura CUDA.

Algoritmo 14: CQEMSO - Inicialização do enxame mestre - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de posição;
            Inicializar os valores de contração-expansão (β) para o enxame mestre;
            Inicializar os valores do o fator de pertubação do melhor global (ω_i) e atribuir ao enxame
            mestre;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i(M));
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g(M));
        end
    end
end
```

O algoritmo 15 mostra o procedimento de busca e otimização envolvendo as partículas originais e o uso de estratégias evolutivas para a geração das réplicas nos enxames escravos do algoritmo CQEMSO sob a arquitetura CUDA.

Algoritmo 15: CQEMSO com fator de inércia (enxames escravos) - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de ponto de inclinação (LIP ou p) (Equação 2.28);
                Atualizar valor de posição da variável j (Equação 2.36);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
                Gerar novo valor para o fator de contração-expansão (β) atravé da mutação
                (Equação 3.23));
                Gerar novo valor para o fator de pertubação atravé da mutação (Equação 2.20));
                for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                    Atualizar valor de ponto de inclinação (LIP ou p) para a réplica k (Equação
                    3.24);
                    Atualizar valor de posição da variável j para a réplica k (Equação 3.25);
                    Aplicar condições de contorno descrita no Algoritmo 5;
                if Houver cópias da partícula com resultados de condições de contorno distintas
                then
                    Selecionar a melhor cópia da partícula e classifica-la como a solução da
                    mesma;
                Avaliar réplica (Fitness);
                Comparar réplica e partícula original;
                if fitness(Replica) melhor do que fitness(Original) then
                    Substituir partícula original pela réplica (Original = Replica);
                    Substituir fitness original por fitness réplica;
            Obter os melhor local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 16 mostra o procedimento de busca e otimização envolvendo as partículas originais e o uso de estratégias evolutivas para a geração das réplicas no enxame mestre do algoritmo CQEMSO sob a arquitetura CUDA.

Algoritmo 16: CQEMSO com fator de inércia (enxame mestre) - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de ponto de inclinação (LIP ou p) (Equação 3.30);
                Atualizar valor de posição da variável j (Equação 3.31);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas
            then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da
                mesma;
            Avaliar partícula (Fitness);
            for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
                Gerar novo valor para o fator de contração-expansão (β) atravé da mutação
                (Equação 3.23));
                Gerar novo valor para o fator de pertubação atravé da mutação (Equação
                2.20));
                for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                    Atualizar valor de ponto de inclinação (LIP ou p) para a réplica k
                    (Figura 3.8);
                    Atualizar valor de posição da variável j para a réplica k (Equação 3.29);
                    Aplicar condições de contorno descrita no Algoritmo 5;
                if Houver cópias da partícula com resultados de condições de contorno
                distintas then
                    Selecionar a melhor cópia da partícula e classifica-la como a solução da
                    mesma;
                Avaliar réplica (Fitness);
                Comparar réplica e partícula original;
                if fitness(Replica) melhor do que fitness(Original) then
                    Substituir partícula original pela réplica (Original = Replica);
                    Substituir fitness original por fitness réplica;
            Obter os melhor local da partícula (P_i(M));
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g(M));
                if P_g(S) for melhor que P_g(M) then
                    Atribuir o valor de P_g(S) a P_g(M);
        end
    end
end
```

O Algoritmo 17 mostra a execução de CQEMSO na íntegra sob a arquitetura CUDA.

Algoritmo 17: Algoritmo CQEMSO em CUDA

Alocar memória para as estruturas dos enxames escravos (posição (X), velocidade (V)) (GPU);
Alocar memória para o melhores valores dos enxames escravos (local (P_i), global (P_g)) (GPU);
Alocar memória para o melhor resultado obtido pelos enxames escravos ($P_{g(S)}$) (CPU e GPU);
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização dos enxames escravos (Algoritmo 13);
Sincronizar GPU;
Inicializar o melhor resultado encontrado pelos enxames escravos ($P_{g(S)}$);
Alocar memória para estruturas do enxame mestre (posição (X), velocidade (V)) (GPU);
Alocar memória para melhores valores dos enxame mestre (local (P_i), global ($P_{g(M)}$)) (GPU);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização do enxame mestre (Algoritmo 14);
Sincronizar GPU;
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
for $iter \leftarrow 1$ **to** *(QUANTIDADE DE ITERAÇÕES*-1 **do**
    Atualizar fator β (Equação 2.32 ou 2.33);
    Atualizar fator α (Equação 2.29 ou 2.30);
    Obter a média dos melhores locais dos enxames escravos (Equação 2.31);
    Executar o *kernel* de busca e otimização enxames escravos (Algoritmo 15);
    Sincronizar GPU;
    Obter o melhor resultado encontrado pelos enxames escravos ($P_{g(S)}$);
    Obter a média dos melhores locais do enxame mestre (Equação 2.31);
    Executar o *kernel* de busca e otimização enxame mestre (Algoritmo 16);
    Sincronizar GPU;
Copiar os valores de melhor global dos enxames escravos para a memória principal (GPU para CPU);
Copiar os valores de melhor global do enxame mestre para a memória principal (GPU para CPU);
Liberar memória;

Capítulo 4

Análise de Desempenho da *Proposta*

No intuito de avaliar os algoritmos propostos e apresentados no capítulo anterior, foram escolhidos quatro problemas de otimização global com restrições voltados para a engenharia, onde todos os problemas são de minimização. Dessa forma, para cada uma dessas categorias foram selecionados alguns problemas para os quais foram aplicados os dois algoritmos propostos: CEMSO e CQEMSO.

É importante observar que os algoritmos CEMSO e CQEMSO foram implementados a princípio, visando apenas a otimização restritiva com ênfase em problemas de engenharia, embora seja também compatível com problemas de otimização não-restritiva e outras classes de problemas de otimização que envolvam funções contínuas.

Antes de apresentar em maiores detalhes os resultados obtidos nas simulações, é necessário abordar trê aspectos importantes na comparação dos resultados, que são:

- Adaptações realizadas nos algoritmos, mais especificamente ao tratamento dado às restrições encontradas nos problemas engenharia;
- Utilizaçã das versões mono-enxame e multi-enxame de cada algoritmo submetido aos testes;
- Parâmetros utilizados nas configurações dos experimentos realizados.

A máquina utilizada para os experimentos foi um Apple MacBook Pro 15' modelo 2012 (9,1). A configuração de hardware possui as seguintes especificações:

1. ***CPU***: Intel Core i7 com quatro núcleos e velocidade de *clock* de 2.66 GHz cada;
2. ***Cache***: L2 (por núcleo) de 256 KB e L3 de 6 MB;
3. ***Memória RAM***: 8 GB de memória RAM DDR3 com velocidade de 1600 MHz;
4. ***GPU***: NVIDIA GeForce GT 650M com 384 núcleos CUDA e velocidade de *clock* de 900 MHz;
5. ***Memória VRAM***: 512 GB de memória VRAM GDDR5;
6. ***Arquitetura da GPU***: Kepler[1];
7. ***Capacidade de Computabilidade***: 3.0;

4.1 Otimização Restritiva em Problemas de Engenharia

Segundo [Teixeira 2012], uma das principais características em problemas de otimização na área de engenharia é a presença de condições restritivas, geralmente associadas ao intervalo que uma determinada variável pode adotar ou a relação existente entre as variáveis, sendo que a não satisfação dessas restrições ocasiona uma violação.

O exemplo abaixo trata de um problema de otimização global cuja a restrição se estende a valores menores ou iguais a zero.

$$\text{Minimizar ou Maximizar } f(X_i) \tag{4.1}$$

Sujeito a:

$$g \leq 0 \tag{4.2}$$

Onde:

- X_i: Valores de parâmetros em uma partícula;
- $f(X_i)$: Valor de *fitness* da solução;
- g: função de restrição.

A quantidade de funções de restrição podem variar de acordo com o problema.

[1]Os algoritmos implementados são compatíveis com versões anteriores da arquitetura CUDA (i.e. Fermi e Tesla), bem como a nova geraçã de placas NVIDIA (série Maxwell)

Penalidades Aplicadas a Resultados com Violações

No intuito de representar os valores de média e variância do resultado obtido em uma execução com maior precisão, aplica-se valores de penalidade a cada solução com violações. É importante ressaltar que os valores somados as soluções foram selecionados arbitráriamente, estando sujeito a outros valores. O procedimento de penalidades é descrito abaixo.

- ***Projeto de Viga de Aço (WBD)***: Caso a solução obtida ao final de uma execução possua violações, o valor da solução de *fitness* será somado pelo valor resultante da multiplicação da quantidade de violações por ***1.200000***.
- ***Projeto de Vaso de Pressão (DPV)***: Caso a solução obtida ao final de uma execução possua violações, o valor da solução de *fitness* será somado pelo valor resultante da multiplicação da quantidade de violações por ***7000.000000***.
- ***Peso da Tensão/Compressão sobre Mola (MWTCS)***: Caso a solução obtida ao final de uma execução possua violações, o valor da solução de *fitness* será somado pelo valor resultante da multiplicação da quantidade de violações por ***0.850000***.
- ***Projeto de Redutor de Velocidade (SRD)***: Caso a solução obtida ao final de uma execução possua violações, o valor da solução de fitness será somado pelo valor resultante da multiplicação da quantidade de violações por ***2200.000000***.

A Equação 4.3 expõe o procedimento descrito nos items acima

$$f(X_i) = \begin{cases} f(X_i) & \text{se } VIO(X_i) = 0 \\ f(X_i) + (\zeta VIO(X_i)) & \text{se } VIO(X_i) > 0 \end{cases} \tag{4.3}$$

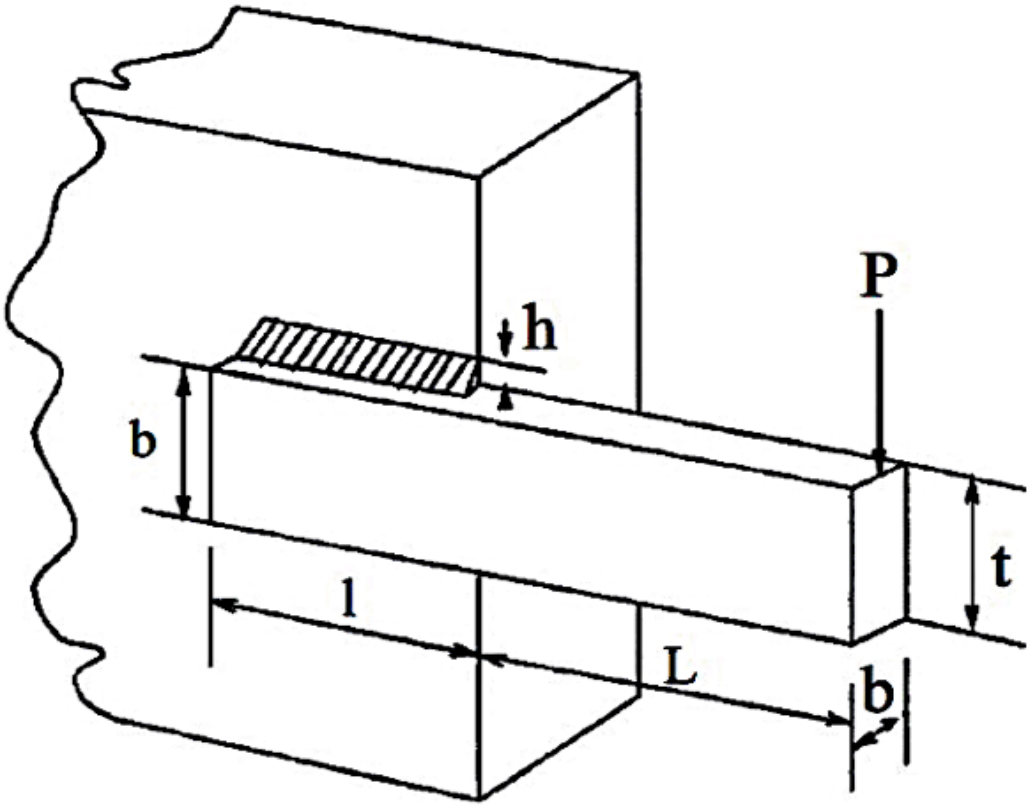
$$\zeta = \begin{cases} 1.200000 & \text{se WBD} \\ 7000.000000 & \text{se DPV} \\ 0.850000 & \text{se MWTCS} \\ 2200.000000 & \text{se SRD} \end{cases} \tag{4.4}$$

- $VIO(X_i)$: Quantidade de violações;
- k: Índice da execução;
- $P_{g(k)}$: Melhor resultado obtido na execução k;
- ζ: Constante de violação;

4.1.1 Projeto de Viga de Aço

Segundo [Teixeira 2012, Belegundu & Arora 1985, Arora 2004], o Projeto de Viga de Aço (WBD) consiste na minimização do custo de fabricação de uma viga de aço, sujeita a algumas restrições, tais como: tensão do cisalhamento, esforços de flexão na viga, flambagem de carga na barra e, deflexão do feixe de extremidade e restrições laterais. As variáveis do problema são: Espessura da solda (h); largura do feixe (l); espessura da viga (t); comprimento da junta soldada (b). A Figura 4.1 mostra o esquema do WBD.

Figura 4.1: Esquema do WBD. Fonte:[Liu 2006]



O WBD pode ser formalizado matematicamente através do seguinte esquema:

$$\text{Minimizar } f(X_i) = 1.10471h^2l + 0.04811tb(14.0+l) \tag{4.5}$$

Sujeito a:

$$g_1 = \tau - \tau_{\max} \leq 0 \tag{4.6}$$

$$g_2 = \sigma - \sigma_{\max} \leq 0 \tag{4.7}$$

$$g_3 = h - b \leq 0 \tag{4.8}$$

$$g_4 = 0.10471h^2 + 0.04811tb(14.0+l) - 5.0 \leq 0 \tag{4.9}$$

$$g_5 = 0.125 - h \leq 0 \tag{4.10}$$

$$g_6 = \delta - \delta_{\max} \leq 0 \tag{4.11}$$

$$g_7 = P - P_c \leq 0 \tag{4.12}$$

Onde:

$$\tau = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{1}{2R} + (\tau'')^2} \tag{4.13}$$

$$\tau' = \frac{P}{\sqrt{2}hl} \tag{4.14}$$

$$\tau'' = \frac{MR}{J} \tag{4.15}$$

$$M = P\left(L\frac{1}{2}\right) \tag{4.16}$$

$$R = \sqrt{\frac{t^2}{4} + \left(\frac{h+t}{2}\right)^2} \tag{4.17}$$

$$J = 2\left\{\sqrt{2}hl\left[\frac{t^2}{12} + \left(\frac{h+t}{2}\right)^2\right]\right\} \tag{4.18}$$

$$\sigma = \frac{6PL}{bt^2} \tag{4.19}$$

$$\delta = \frac{4PL^3}{Eb^2} \tag{4.20}$$

$$P_c = \frac{4.013E\sqrt{\frac{T^2b^6}{36}}}{L^2} + \left(1 - \frac{t}{2L}\sqrt{\frac{E}{4G}}\right) \tag{4.21}$$

Para as constantes P, L,E, G, τ, $\tau_{\max}$, $\sigma_{\max}$, $\delta_{\max}$, os valores são: P=6000 lb, L=14 in, E=$30x10^6$ psi, G=$12x10^6$ psi, $\tau_{\max}$=13600 psi, $\sigma_{\max}$=30000 psi, $\delta_{\max}$=0.25 in.

Os limites de variáveis h, l, t, b são: $0.1 \leq h \leq 2.0$; $0.1 \leq l \leq 10.0$; $0.1 \leq t \leq 10.0$; $0.1 \leq b \leq 2.0$.

No intuito de avaliar os algoritmos CEMSO e CQEMSO, foram utilizadas as seguintes configurações: Iterações = 20, 40, 80, 200, 400 e 1000; quantidade de execuções = 500; quantidade de partículas para cada enxame escravo = 80; quantidade de réplicas por partícula = 4; quantidade de enxames escravos = 4; constante de pertubação para o melhor global (σ_g) = 0.005; parâmetro de estratégias para a mutação dos fatores de inércia e de aceleração (σ) = 0.22; fatores C_1 e C_2 (COEMSO e CEMSO) = 2.05; fator de aceleração para o enxame mestre (C_3) = 2.02; quantidade de testes executados para cada algoritmo = 500; Fator de diminuição de velocidade (δ) = diminuir a velocidade a 40% (0.4); Fator de penalidade para resultados com violações = 200; Constante de probabilidade (θ) = 0.5; Velocidade mínima (V_{MIN}) = metade do valor de limite máximo de posição com sinal ou direção invertida ($-(\frac{X_{MAX}}{2})$). Os valores de fator de inércia (w) (CEMSO), bem como fatores α e β (CQEMSO e COQMSO) são gerados a cada iteração pelas Equações 2.2, 2.30 e 2.32 (decrescimento randômico).

Além dos algoritmos CEMSO e CQEMSO, foram utilizados para efeito de comparação os algoritmos PSO, EPSO, QPSO, PSO+EE e QPSO+EE baseados na implementação exposta nos capítulos 3 e 4, bem como suas versões multi-enxame (COMSO, COEMSO e COQMSO). As Tabelas 4.1, 4.2, 4.3, 4.4, 4.5 e 4.6 mostram os resultados obtidos nos experimentos realizados para o problema WBD.

Tabela 4.1: Comparação dos resultados para 20 iterações para o WBD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Média de Tempo
PSO	1.8505935	1.463022	34	$2.564556e-01$	0.003948 s
EPSO	1.5684304	1.463979	0	$1.336573e-02$	0.008738 s
QPSO	1.8282308	1.486162	0	$6.244039e-02$	0.005086 s
PSO+EE	1.4982081	1.461114	15	$2.894184e-02$	0.010330 s
QPSO+EE	1.5643358	1.477839	0	$8.109181e-03$	0.012069 s
COMSO	1.5603076	1.463022	0	$4.005055e-02$	0.007105 s
COEMSO	1.4836584	1.461930	0	$2.048346e-04$	0.008738 s
COQMSO	1.5940962	1.480159	0	$7.410453e-03$	0.007331 s
CEMSO	1.4663554	1.459617	0	$2.038871e-05$	0.020186 s
CQEMSO	1.4942197	1.466900	0	$2.435805e-04$	0.023278 s

Tabela 4.2: Comparação dos resultados para 40 iterações para o WBD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Média de Tempo
PSO	1.6840292	1.459761	13	$1.656449e-01$	0.007885 s
EPSO	1.4998146	1.461595	0	$1.658351e-03$	0.019432 s
QPSO	1.6928597	1.475570	0	$4.738354e-02$	0.006457 s
PSO+EE	1.4664732	1.459060	12	$1.974745e-02$	0.018670 s
QPSO+EE	1.4866591	1.464289	0	$5.104804e-04$	0.019807 s
COMSO	1.5006838	1.459365	0	$1.598162e-02$	0.014208 s
COEMSO	1.4689180	1.460223	0	$3.498366e-05$	0.040586 s
COQMSO	1.5096221	1.466539	0	$1.238052e-03$	0.014193 s
CEMSO	1.4608096	1.459016	0	$2.711479e-06$	0.040564 s
CQEMSO	1.4709755	1.461475	0	$2.777896e-05$	0.046365 s

Tabela 4.3: Comparação dos resultados para 80 iterações para o WBD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Média de Tempo
PSO	1.6091560	1.459005	18	$1.250674e-01$	0.015634 s
EPSO	1.4836838	1.459378	0	$2.093278e-03$	0.039561 s
QPSO	1.5818164	1.465010	0	$2.096631e-02$	0.012272 s
PSO+EE	1.4633118	1.458917	12	$2.141930e-02$	0.041377 s
QPSO+EE	1.4674986	1.459548	0	$2.977842e-05$	0.039491 s
COMSO	1.4862352	1.458899	0	$1.592304e-02$	0.025823 s
COEMSO	1.4634700	1.459063	0	$1.047958e-05$	0.080989 s
COQMSO	1.4781467	1.463547	0	$1.227649e-04$	0.028836 s
CEMSO	1.4590505	1.458884	0	$2.501800e-07$	0.079491 s
CQEMSO	1.4628793	1.459429	0	$4.014850e-06$	0.093460 s

Tabela 4.4: Comparação dos resultados para 200 iterações para o WBD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Média de Tempo
PSO	1.5708290	1.458884	15	$7.606745e-02$	0.029750 s
EPSO	1.4724385	1.459018	0	$5.815839e-04$	0.123378 s
QPSO	1.4954505	1.460265	0	$3.283378e-03$	0.031135 s
PSO+EE	1.4599162	1.458883	6	$9.453537e-03$	0.098754 s
QPSO+EE	1.4612885	1.459001	0	$2.532672e-06$	0.138733 s
COMSO	1.4737913	1.458883	0	$8.758966e-03$	0.065192 s
COEMSO	1.4610219	1.458934	0	$5.525516e-06$	0.194564 s
COQMSO	1.4665189	1.459900	0	$1.345914e-05$	0.069501 s
CEMSO	1.4589198	1.458883	0	$4.774822e-08$	0.199794 s
CQEMSO	1.4598476	1.458947	0	$3.984075e-07$	0.235843 s

Tabela 4.5: Comparação dos resultados para 400 iterações para o WBD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Média de Tempo
PSO	1.5483888	1.458883	16	$7.169842e-02$	0.071052 s
EPSO	1.4721707	1.459051	0	$9.933746e-04$	0.262710 s
QPSO	1.4721377	1.459977	0	$3.231913e-04$	0.064524 s
PSO+EE	1.4590291	1.458883	7	$2.021572e-02$	0.281344 s
QPSO+EE	1.4600829	1.458931	0	$6.933031e-07$	0.210510 s
COMSO	1.4707952	1.458883	0	$6.995738e-03$	0.128111 s
COEMSO	1.4606092	1.458933	0	$5.313210e-06$	0.392295 s
COQMSO	1.4629492	1.459159	0	$3.797244e-06$	0.144073 s
CEMSO	1.4588925	1.458883	0	$6.605261e-09$	0.434279 s
CQEMSO	1.4592775	1.458905	0	$9.413310e-08$	0.480656 s

Tabela 4.6: Comparação dos resultados para 1000 iterações para o WBD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Média de Tempo
PSO	1.4848748	1.458883	16	$2.482875e-02$	0.191308 s
EPSO	1.4661679	1.458955	0	$3.744075e-04$	0.661921 s
QPSO	1.4642436	1.459244	0	$3.121671e-05$	0.173542 s
PSO+EE	1.4589864	1.458883	8	$1.448484e-02$	0.658238 s
QPSO+EE	1.4593910	1.458890	0	$1.715943e-07$	0.625464 s
COMSO	1.4615798	1.458883	0	$1.098289e-03$	0.327262 s
COEMSO	1.4600592	1.458891	0	$2.481045e-06$	0.661921 s
COQMSO	1.4607121	1.459244	0	$9.671503e-07$	0.353992 s
CEMSO	1.4588897	1.458883	0	$4.109770e-10$	0.983641 s
CQEMSO	1.4590260	1.458890	0	$1.288738e-08$	1.083012 s

As Figuras 4.2 e 4.3 mostram, respectivamente, o comparativo entre os melhores valores obtidos e a média dos resultados encontrados de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.2: Gráfico comparativo dos melhores resutados obtidos para o WBD

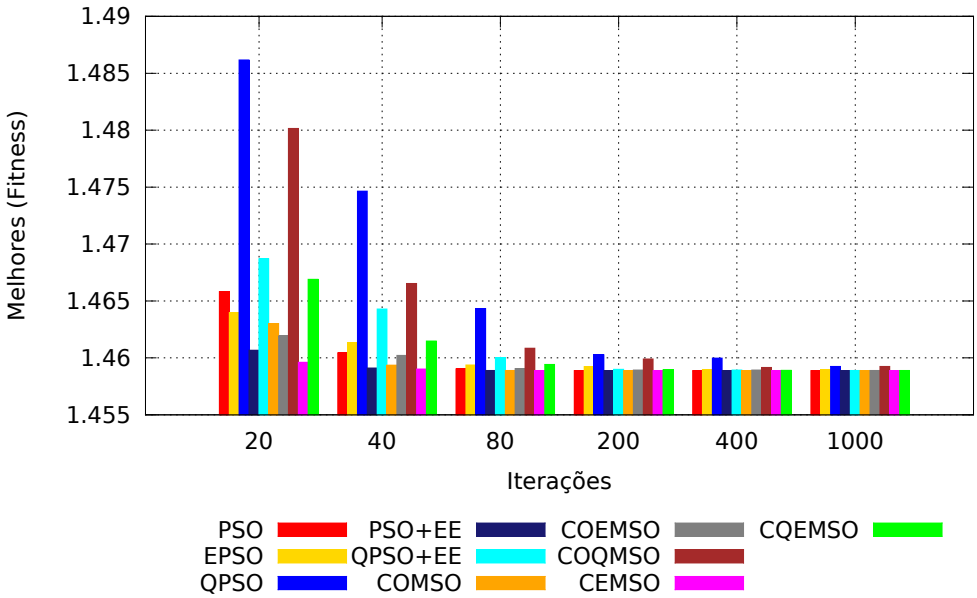
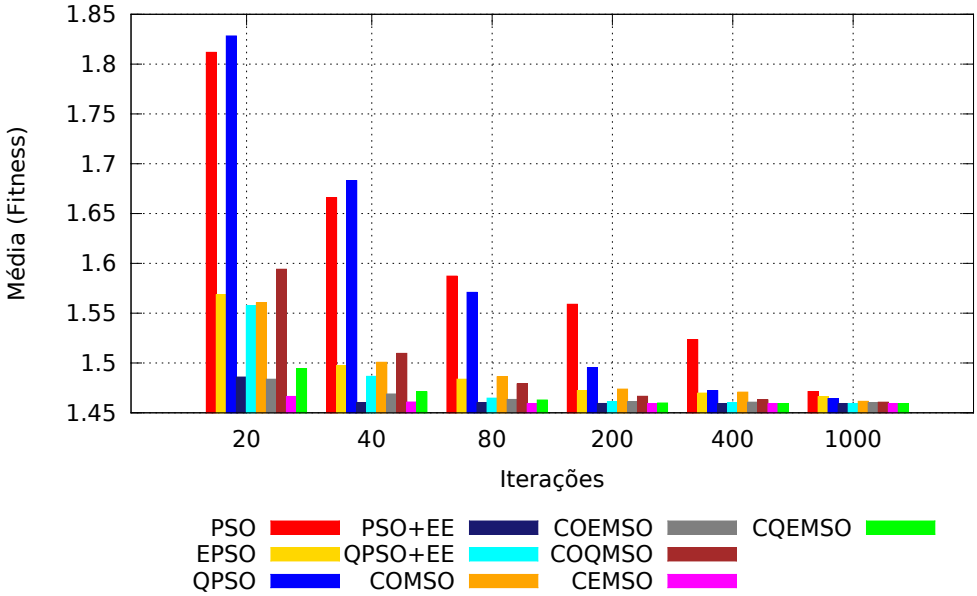


Figura 4.3: Gráfico comparativo das médias para o WBD



As Figuras 4.4 e 4.5 mostram, respectivamente, a varância dos resultados encontrados e o tempo de execução de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.4: Gráfico comparativo dos valores de variância para o WBD

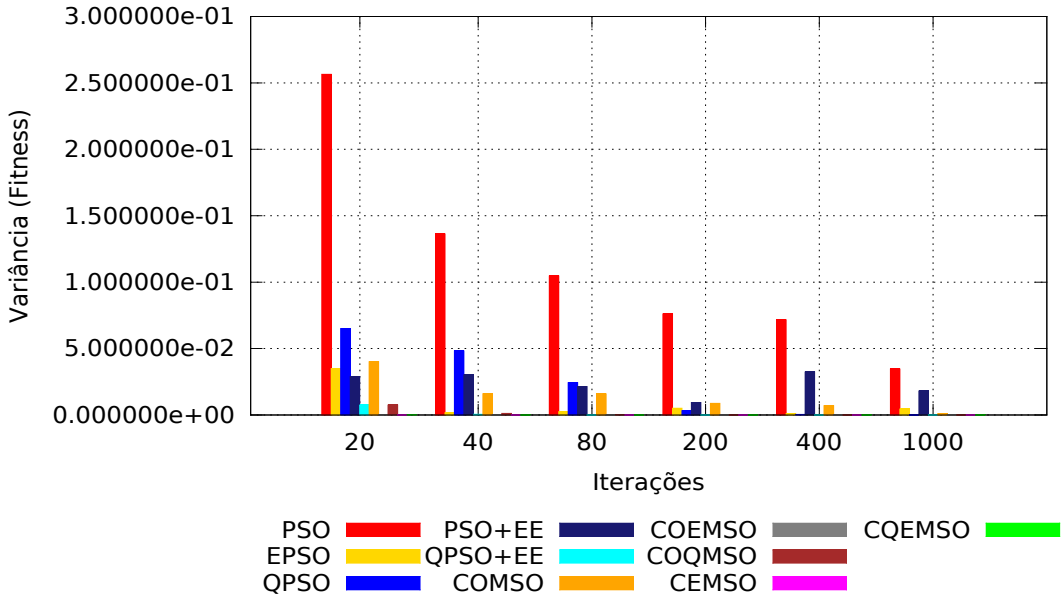
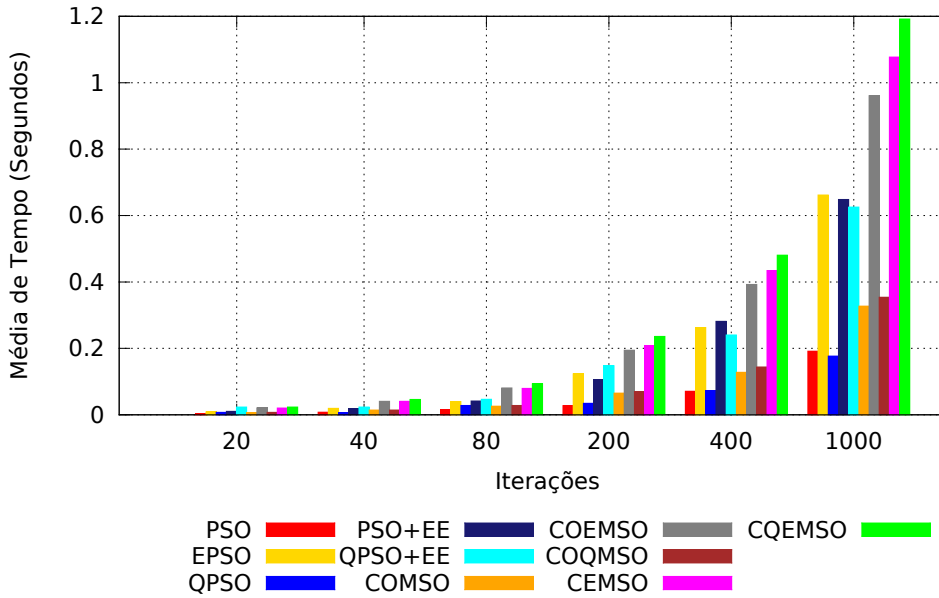


Figura 4.5: Gráfico comparativo da média do tempo de execução dos algoritmos testados (WBD)



As Figuras 4.6 e 4.7 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.6: Convergência dos enxames escravos do algoritmo COMSO para o WBD

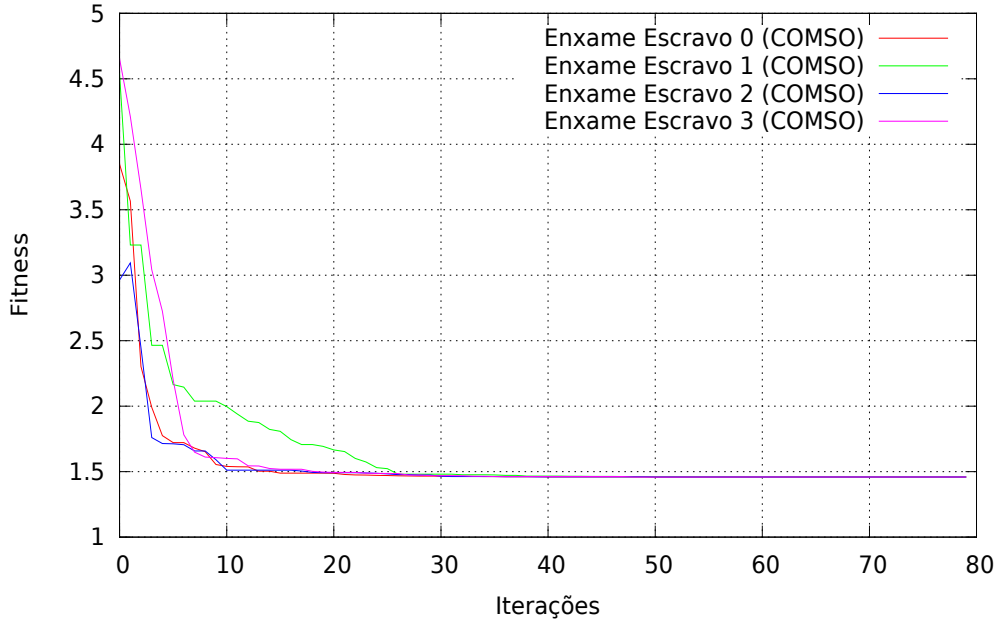
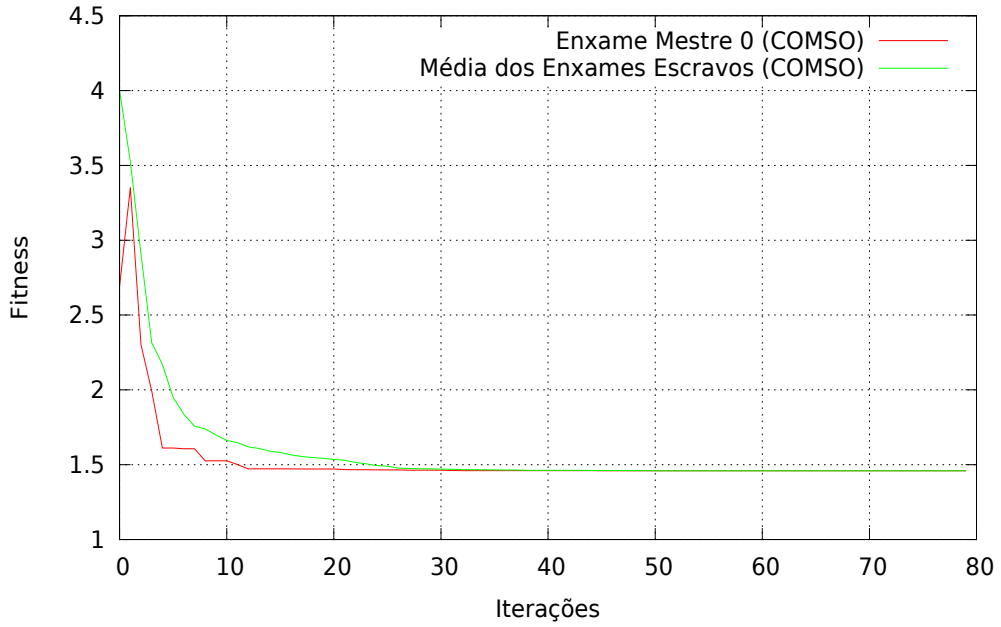


Figura 4.7: Convergência do enxame mestre do algoritmo COMSO para o WBD



As Figuras 4.8 e 4.9 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.8: Convergência dos enxames escravos do algoritmo COEMSO para o WBD

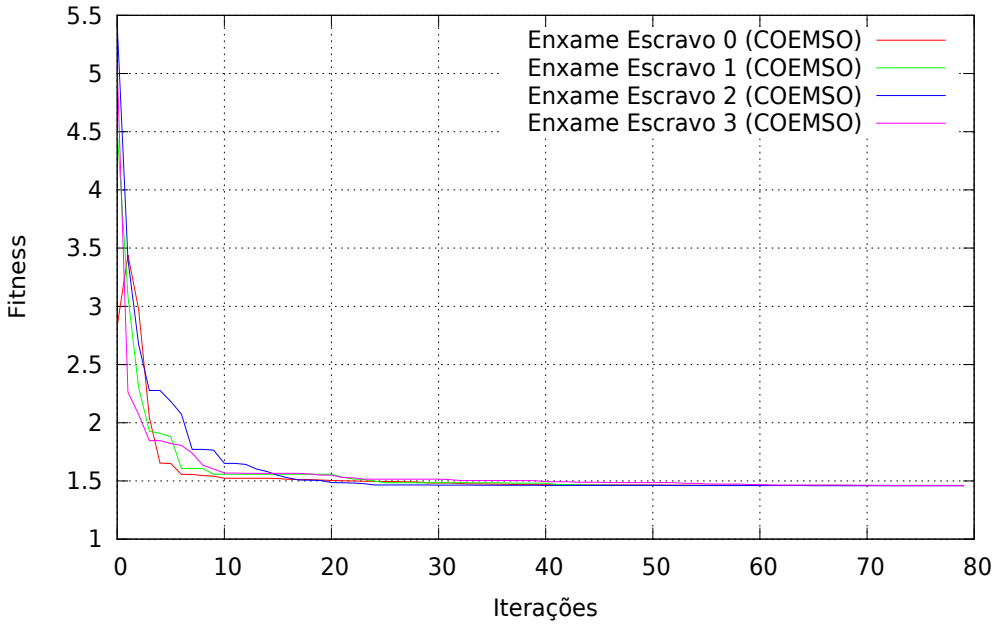
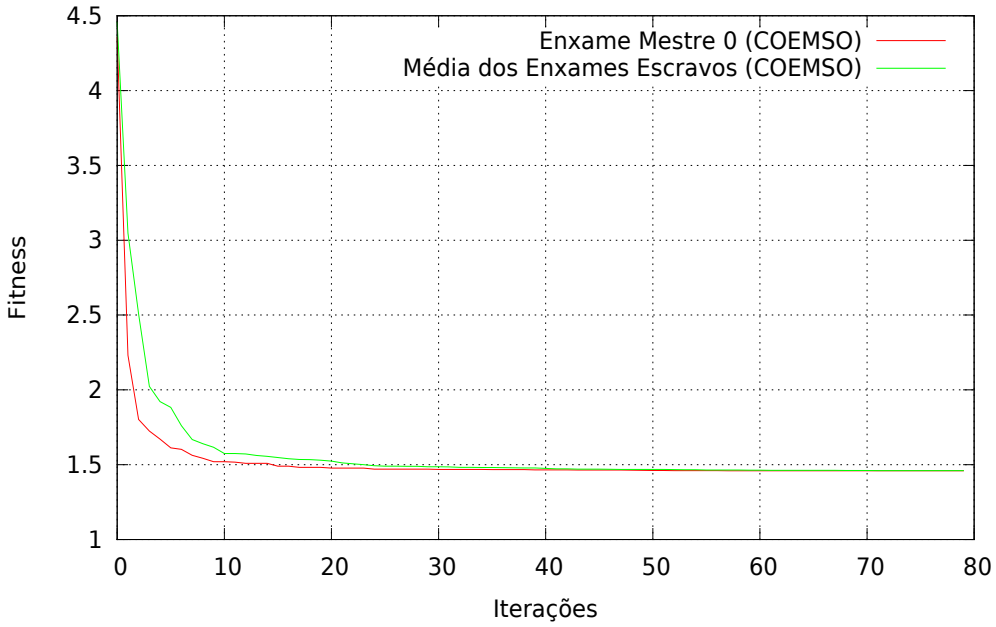


Figura 4.9: Convergência do enxame mestre do algoritmo COEMSO para o WBD



As Figuras 4.10 e 4.11 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COQMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.10: Convergência dos enxames escravos do algoritmo COQMSO para o WBD

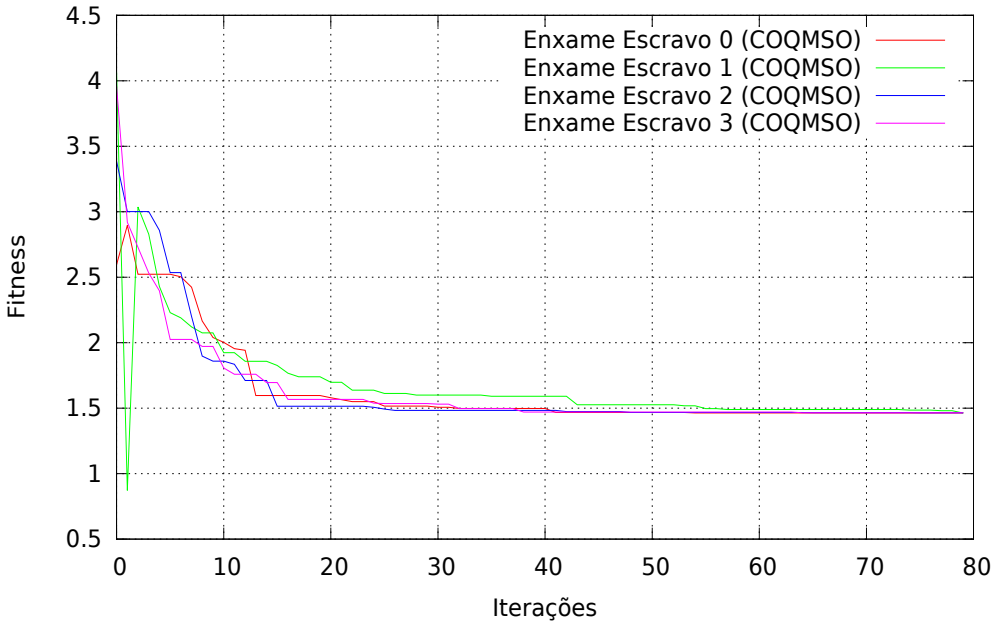
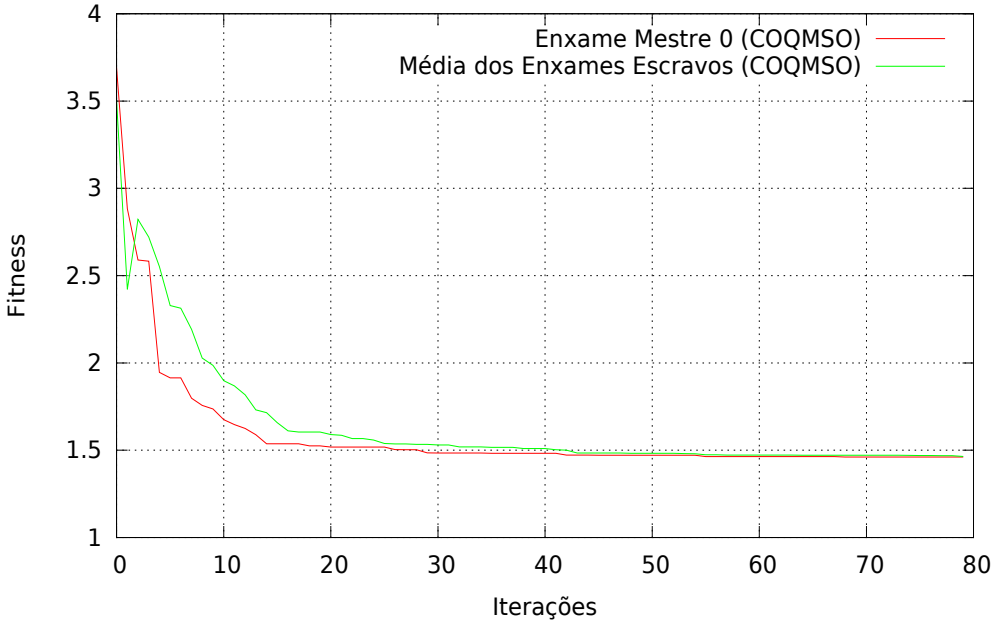


Figura 4.11: Convergência do enxame mestre do algoritmo COQMSO para o WBD



As Figuras 4.12 e 4.13 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.12: Convergência dos enxames escravos do algoritmo CEMSO para o WBD

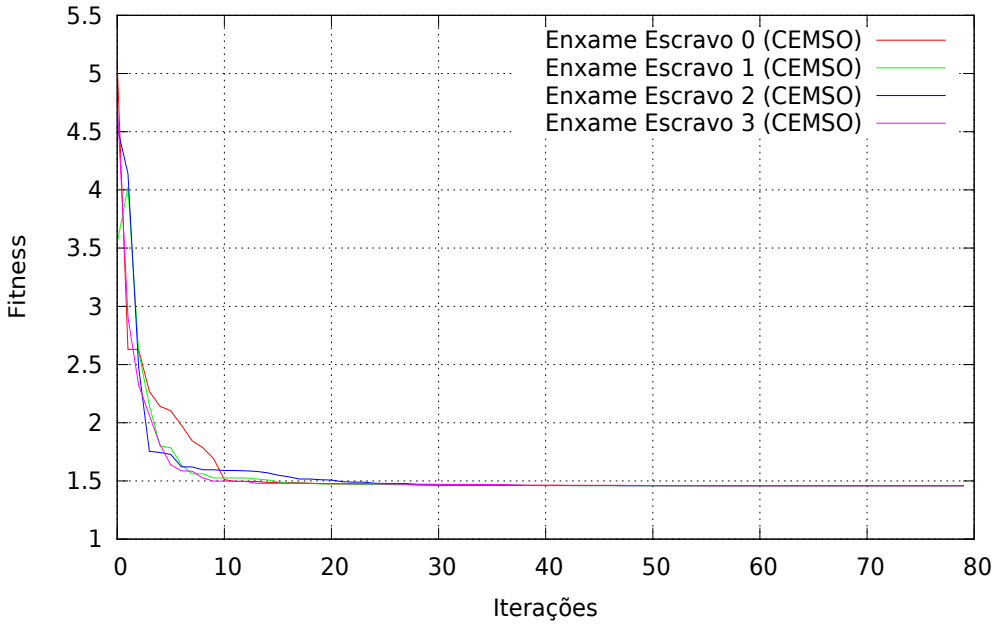
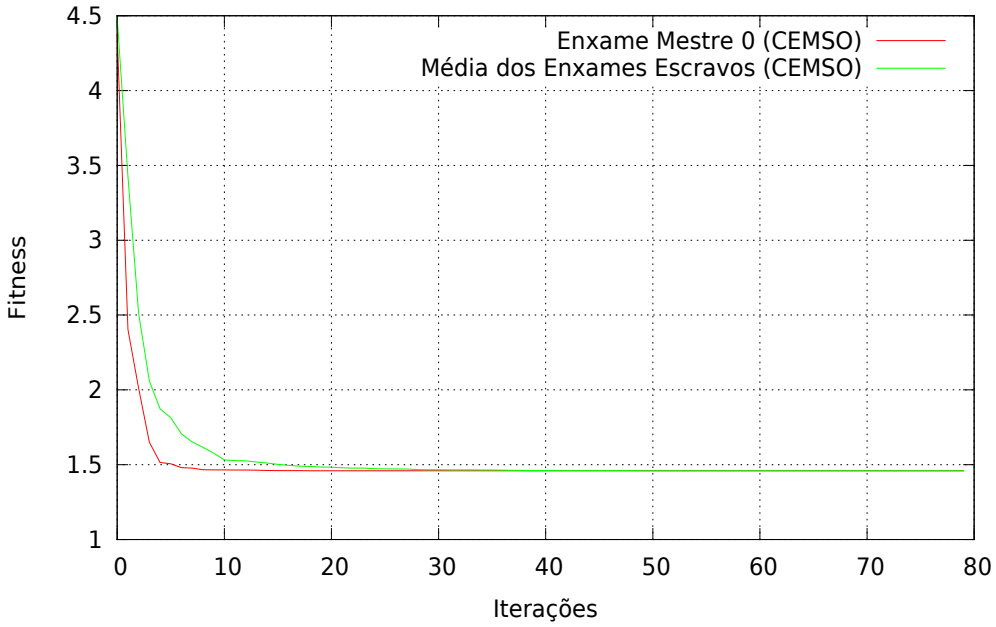


Figura 4.13: Convergência do enxame mestre do algoritmo CEMSO para o WBD



As Figuras 4.14 e 4.15 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.14: Convergência dos enxames escravos do algoritmo CQEMSO para o WBD

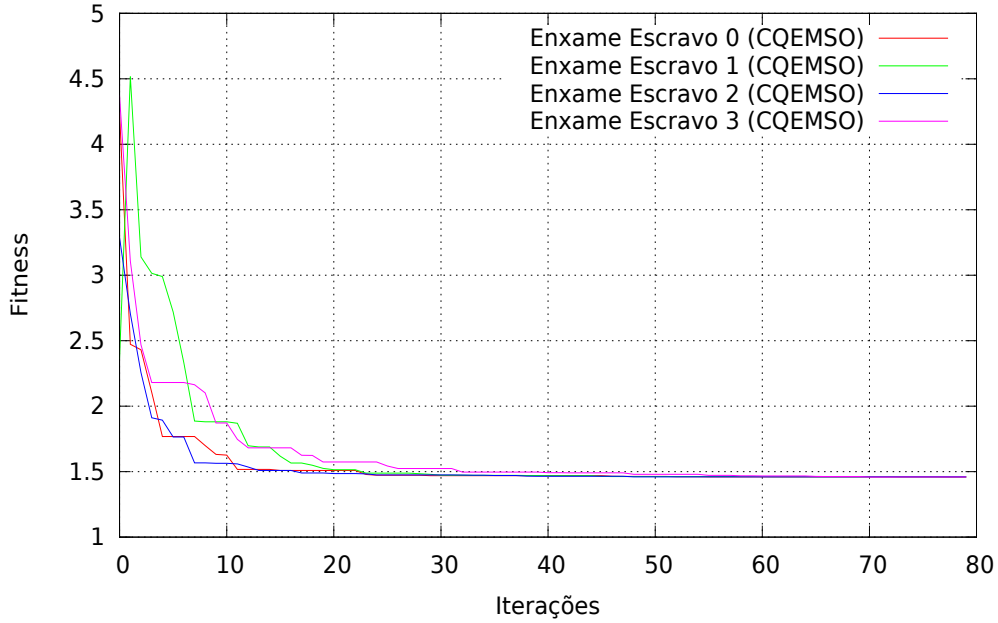
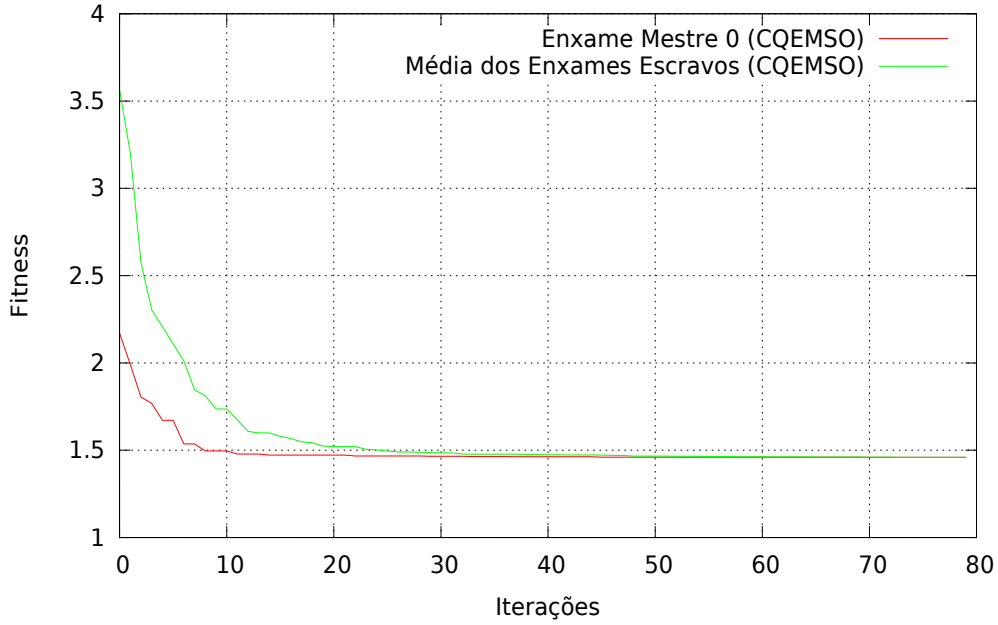


Figura 4.15: Convergência do enxame mestre do algoritmo CQEMSO para o WBD



As Figuras 4.16 e 4.17 mostram a média de convergência dos enxames escravos dos algoritmos COMSO, COEMSO, CEMSO, COQMSO e CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.16: Média de convergência dos enxames escravos para os algoritmo COMSO, CEMSO e COEMSO no problema WBD

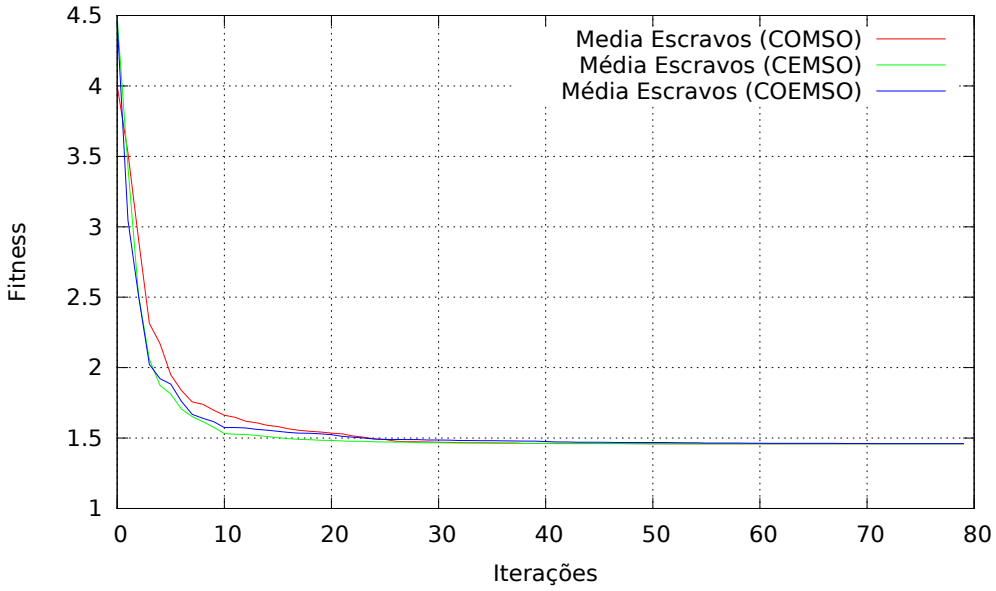
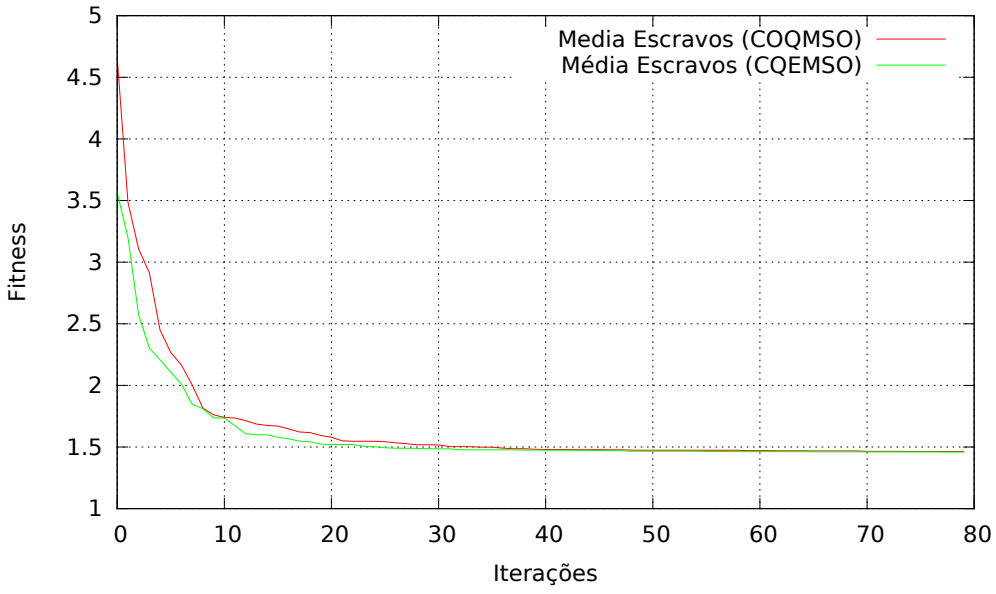


Figura 4.17: Média de convergência dos enxames escravos para os algoritmo COQMSO e CQEMSO no problema WBD



Análise dos Resultados para o WBD

Com os resultados obtidos pelos algoritmos CEMSO e CQEMSO, é possível estabelecer uma comparação dos resultados obtidos nessas simulações com resultados presentes na literatura. São utilizados os resultados do melhor *fitness* obtido através do experimento de 400 iterações.

Segue-se os algoritmos usados para comparação:

- ***Alg. 1 [Teixeira 2012]***: Algoritmo genético com interação social baseado na teoria dos jogos (SIGA);
- ***Alg. 2 [He & Wang 2007]***: Otimização por enxame de partículas com mecanismos co-evolutivos aplicados em um fator de penalidade para otimização restritiva;
- ***Alg. 3 [Coello & Montes 2002]***: Algoritmo genético com mecanismo de manipulação de restrição.
- ***Alg. 4 [Liu 2006]***: Controlador Fuzzy com derivação proporcional adaptado para problemas de engenharia. Os resultados obtidos por [Liu 2006] aplicam uma tolerância de até 0.01% em relação ao valor mínimo de restrição).

Tabela 4.7: Comparação dos resultados para o problema WBD

Variaveis	CEMSO	CQEMSO	Alg. 1	Alg. 2	Alg. 3	Alg. 4
$X_1(h)$	0.205730	0.205727	0.171937	0.202369	0.205986	0.2326
$X_2(l)$	1.517675	1.517707	4.122129	3.544214	3.471328	3.1795
$X_3(t)$	9.036624	9.036655	9.587429	9.048210	9.020224	8,4355
$X_4(b)$	0.205730	0.205729	0.183010	0.205723	0.206480	2.3609
G_1	-0.000977	-0.106445	-8.067400	-12.839796	-0.074092	0.0001[1]
G_2	-0.001953	-0.189453	-39.336800	-1.247467	-0.266227	0.000049[1]
G_3	0.000000	-0.000003	-0.011070	-0.001498	-0.000495	-0.0144
G_4	-3.607646	-3.607639	-3.467150	-3.429347	-3.430043	-0.6697
G_5	-0.080730	-0.080727	-0.236390	-0.079381	-0.080986	-0.4627
G_6	-0.235540	-0.235540	-16.024300	-0.235536	-0.235514	-0.9380
G_7	-0.000488	-0.001465	-0.046940	-11.681355	-58.666440	-0.3069
Violações	0	0	0	0	0	2
Tempo	0.983641	1.083012	N/A	N/A	N/A	N/A
Fitness	1.458883	1.458890	1.664373	1.728024	1.728226	1.8362

- O melhor resultado encontrado pelos algoritmos com base no PSO para as partículas originais foi 1.458883 e foram obtidos pelos seguintes algoritmos:

    1. CEMSO: Obteve o melhor resultado a partir de 200 iterações;
    2. COMSO: Obteve o melhor resultado a partir de 200 iterações;
    3. PSO: Obteve o melhor resultado a partir de 400 iterações;
    4. PSO+EE: Obteve o melhor resultado a partir de 200 iterações.

- O melhor resultado encontrado pelos algoritmos com base no EPSO para as partículas originais foi 1.458891 e foram obtidos pelos seguintes algoritmos:

    1. COEMSO: Obteve o melhor resultado a partir de 1000 iterações;
    2. EPSO: Melhor resultado obtido for 1.458955 (1000 iteração).

- O melhor resultado encontrado pelos algoritmos com base no QPSO para as partículas originais foi 1.458890 e foram obtidos pelos seguintes algoritmos:

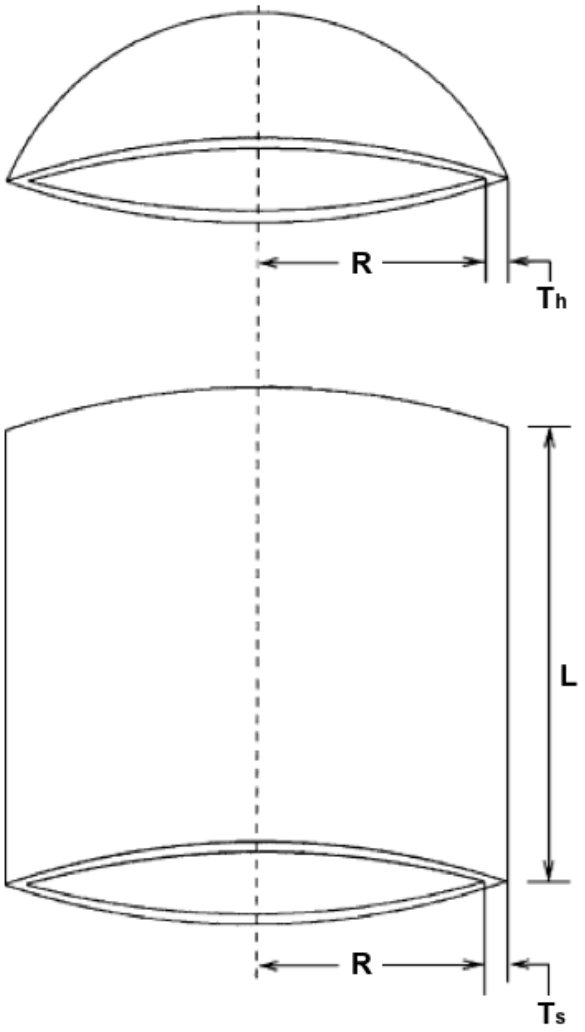
    1. CQEMSO: Obteve o melhor resultado a partir de 1000 iterações;
    2. COQMSO: Melhor resultado obtido for 1.459244 (1000 iteração);
    3. QPSO: Melhor resultado obtido for 1.459244 (1000 iteração);
    4. QPSO+EE: Obteve o melhor resultado a partir de 1000 iterações.

- Os algoritmos que utilizam a abordagem multi-enxame de topologia mestre-escravos e baseados no PSO, EPSO e PSO+EE obtiveram os melhores resultados em todos os experimentos realizados em relação as suas respectivas versões de um enxame;
- Com exceção dos resultados obtidos nos testes envolvendo 1000 iterações, os algoritmos COQMSO e CQEMSO obtiveram os melhores resultados em todos os experimentos realizados em relaçãa as suas respectivas versões de um enxame;
- O algoritmo CEMSO apresentou os melhores resultados em termos de média, variância e melhores resultados em todos os experimentos;
- Os algoritmos CEMSO e CQEMSO obtiveram uma média de convergência superior dentre as soluções comparadas;
- Pode-se observar que a diferença de tempo entre os algoritmos que utilizam a topologia multi-enxame é relativamente pequena. Entretanto, os algoritmos que utilizam os mecanismos de estratégias evolutivas (EPSO, COEMSO, CEMSO e CQEMSO) apresentaram um tempo de execução maior;
- Comparados com os resultados de outras soluções encontradas na literatura (Tabela 4.7), os algoritmos CEMSO e CQEMSO obtiveram os melhores resultados.

4.1.2 Projeto de Vaso de Pressão

Segundo [Cagnina et al. 2008], o Projeto de Vaso de Pressão (DPV) consiste em um problema de engenharia onde busca-se a minimização do custo total, incluindo o custo do material, modelagem e soldagem, de um recipiente cilíndrico, que é limitado em suas extremidades por cabeças hemisféricas. As variáveis do problema são: Espessura da casca (T_s), espessura da cabeça (T_h), raio interno (R) e comprimento da seção cilíndrica do recipiente (L), sendo que esta última variável não inclui as dimensões da cabeça, como exposto na Figura 4.18.

Figura 4.18: Esquema do *Design of a Pressure Vessel* (DPV). Fonte:[Cagnina et al. 2008]



O DPV pode ser formalizado matematicamente através do seguinte esquema:

$$\text{Minimizar } f(X_i) = 0.6224T_sRL + 1.7781T_hR^2 + 3.1661T_s^2L + 19.84T_s^2R \tag{4.22}$$

Sujeito a:

$$g_1 = -T_s + 0.0193R \leq 0 \tag{4.23}$$

$$g_2 = -T_h + 0.00954R \leq 0 \tag{4.24}$$

$$g_3 = \pi R^2 L - \frac{4}{3}\pi R^3 + 1296000 \leq 0 \tag{4.25}$$

$$g_4 = L - 240 \leq 0 \tag{4.26}$$

Os limites das variáveis T_s, T_h, R e L são: $1x0.0625 \leq T_s \leq 99x0.0625$; $1x0.0625 \leq T_h \leq 99x0.0625$; $10 \leq R \leq 200$; $10 \leq L \leq 200$.

No intuito de avaliar os algoritmos CEMSO e CQEMSO, foram utilizadas as seguintes configurações: Iterações = 20, 40, 80, 200, 400 e 1000; quantidade de execuções = 500; quantidade de partículas para cada enxame escravo = 80; quantidade de réplicas por partícula = 4; quantidade de enxames escravos = 4; constante de pertubação para o melhor global (σ_g) = 0.005; parâmetro de estratégias para a mutação dos fatores de inércia e de aceleração (σ) = 0.22; fatores C_1 e C_2 (COEMSO e CEMSO) = 2.05; fator de aceleração para o enxame mestre (C_3) = 2.02; quantidade de testes executados para cada algoritmo = 500; Fator de diminuição de velocidade (δ) = diminuir a velocidade a 40% (0.4); Fator de penalidade para resultados com violações = 200; Constante de probabilidade (θ) = 0.5; Velocidade mínima (V_{MIN}) = metade do valor de limite máximo de posição com sinal ou direção invertida ($-(\frac{X_{MAX}}{2})$). Os valores de fator de inércia (w) (CEMSO), bem como fatores α e β (CQEMSO e COQMSO) são gerados a cada iteração pelas Equações 2.2, 2.30 e 2.32 (decrescimento randômico).

Além dos algoritmos CEMSO e CQEMSO, foram utilizados para efeito de comparação os algoritmos PSO, EPSO, QPSO, PSO+EE e QPSO+EE baseados na implementação exposta nos capítulos 3 e 4, bem como suas versões multi-enxame (COMSO, COEMSO e COQMSO). As Tabelas 4.8, 4.9, 4.10, 4.11, 4.12 e 4.13 mostram os resultados obtidos nos experimentos realizados para o problema DPV.

Tabela 4.8: Comparação dos resultados para 20 iterações para o DPV

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	6731.4162041	5982.066895	21	$2.240954e+06$	0.002386 s
EPSO	6618.5253745	6012.195801	7	$7.552277e+05$	0.006660 s
QPSO	30718.2720000	8666.541016	0	$6.507960e+08$	0.004266 s
PSO+EE	6323.9437861	5901.565430	6	$6.043656e+05$	0.009229 s
QPSO+EE	26007.0593525	6172.185059	0	$1.781614e+08$	0.008568 s
COMSO	6304.4912666	5924.434570	0	$5.422126e+04$	0.005491 s
COEMSO	6230.3268203	5923.743164	0	$2.871320e+04$	0.015158 s
COQMSO	16157.7175391	6734.810059	0	$4.692112e+07$	0.006184 s
CEMSO	6040.8556123	5894.499023	0	$2.045754e+04$	0.015077 s
CQEMSO	15467.9923818	6172.185059	0	$3.367880e+07$	0.018258 s

Tabela 4.9: Comparação dos resultados para 40 iterações para o DPV

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	6817.2191313	5887.469238	9	$3.178447e+06$	0.006543 s
EPSO	6202.4753174	5918.754395	14	$1.150331e+06$	0.014536 s
QPSO	30284.4596650	7029.556152	0	$3.632190e+08$	0.006790 s
PSO+EE	6176.6267168	5887.009277	9	$7.969057e+05$	0.014057 s
QPSO+EE	19368.2563789	7007.216797	0	$1.045615e+08$	0.019444 s
COMSO	6157.1232217	5887.469238	0	$9.353219e+04$	0.011113 s
COEMSO	6080.8916016	5899.939453	0	$2.107476e+04$	0.027367 s
COQMSO	17213.9750127	7029.556152	0	$4.517129e+07$	0.012418 s
CEMSO	5991.5440498	5886.404785	0	$1.697652e+04$	0.030289 s
CQEMSO	11694.4727549	6310.440918	0	$1.908943e+07$	0.037686 s

Tabela 4.10: Comparação dos resultados para 80 iterações para o DPV

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	6544.9533076	5885.587402	11	$1.936918e+06$	0.012952 s
EPSO	6093.8568003	5897.283691	9	$7.341586e+05$	0.039952 s
QPSO	26325.2965664	6718.559570	0	$2.198838e+08$	0.016451 s
PSO+EE	6100.6447783	5885.561035	7	$5.822786e+05$	0.035889 s
QPSO+EE	8929.5433223	6054.459961	0	$1.171073e+07$	0.047808 s
COMSO	6097.4438047	5885.386230	0	$6.388287e+04$	0.021428 s
COEMSO	5992.7744785	5889.538574	0	$1.164101e+04$	0.053609 s
COQMSO	15424.0089678	6718.559570	0	$2.589068e+07$	0.023942 s
CEMSO	5958.4107666	5885.349121	0	$1.241268e+04$	0.049225 s
CQEMSO	7104.0284355	5988.482910	0	$8.409803e+05$	0.071020 s

Tabela 4.11: Comparação dos resultados para 200 iterações para o DPV

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	6487.1932280	5885.393555	8	$2.182874e+06$	0.026495 s
EPSO	6031.5286919	5888.115723	8	$6.409043e+05$	0.064113 s
QPSO	18287.9287891	7604.359375	0	$8.075297e+07$	0.035135 s
PSO+EE	5954.0258770	5885.358398	8	$6.752760e+05$	0.068479 s
QPSO+EE	6080.3913965	5887.322266	0	$2.510998e+04$	0.092996 s
COMSO	6050.9174619	5885.338867	0	$3.824481e+04$	0.053617 s
COEMSO	6129.4582207	5947.567871	0	$2.545555e+04$	0.070999 s
COQMSO	12005.8414121	7093.578125	0	$6.707178e+06$	0.093548 s
CEMSO	5920.3159521	5885.335938	0	$6.079593e+03$	0.152882 s
CQEMSO	5964.5681787	5886.697266	0	$4.410151e+03$	0.098930 s

Tabela 4.12: Comparação dos resultados para 400 iterações para o DPV

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	6491.8059214	5885.336914	9	$2.004652e+06$	0.046836 s
EPSO	5998.5162314	5885.584961	9	$7.013046e+05$	0.256503 s
QPSO	11978.7127402	6392.862305	0	$2.992412e+07$	0.072544 s
PSO+EE	5905.8116514	5885.335938	6	$4.359305e+05$	0.202876 s
QPSO+EE	5939.9983496	5885.776367	0	$2.953556e+03$	0.236373 s
COMSO	6026.8877285	5885.333984	0	$2.827256e+04$	0.103889 s
COEMSO	5934.3129883	5885.584961	0	$6.611149e+03$	0.254855 s
COQMSO	8955.8524502	6392.862305	0	$3.428893e+06$	0.125640 s
CEMSO	5896.4997539	5885.333984	0	$1.127099e+03$	0.306648 s
CQEMSO	5898.0259609	5885.344727	0	$1.976792e+02$	0.351175 s

Tabela 4.13: Comparação dos resultados para 1000 iterações para o DPV

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	6333.1286323	5885.333984	8	$8.100261e+05$	0.118589 s
EPSO	5986.0282939	5885.336914	8	$6.188412e+05$	0.395274 s
QPSO	7634.2738311	5979.822266	0	$3.036323e+06$	0.136551 s
PSO+EE	5889.4266943	5885.333984	6	$4.090690e+05$	0.512955 s
QPSO+EE	5909.0939063	5885.356934	0	$1.061649e+03$	0.472782 s
COMSO	6004.8071768	5885.333984	0	$2.936062e+04$	0.255995 s
COEMSO	5932.9511133	5885.336914	0	$7.146272e+03$	0.637615 s
COQMSO	6869.2100430	5979.822266	0	$7.628519e+05$	0.295996 s
CEMSO	5886.6388926	5885.333984	0	$1.277193e+01$	0.728123 s
CQEMSO	5888.1475645	5885.334961	0	$1.574957e+01$	0.945307 s

As Figuras 4.19 e 4.20 mostram, respectivamente, o comparativo entre os melhores valores obtidos e a média dos resultados encontrados de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.19: Gráfico comparativo dos melhores resutados obtidos para o DPV

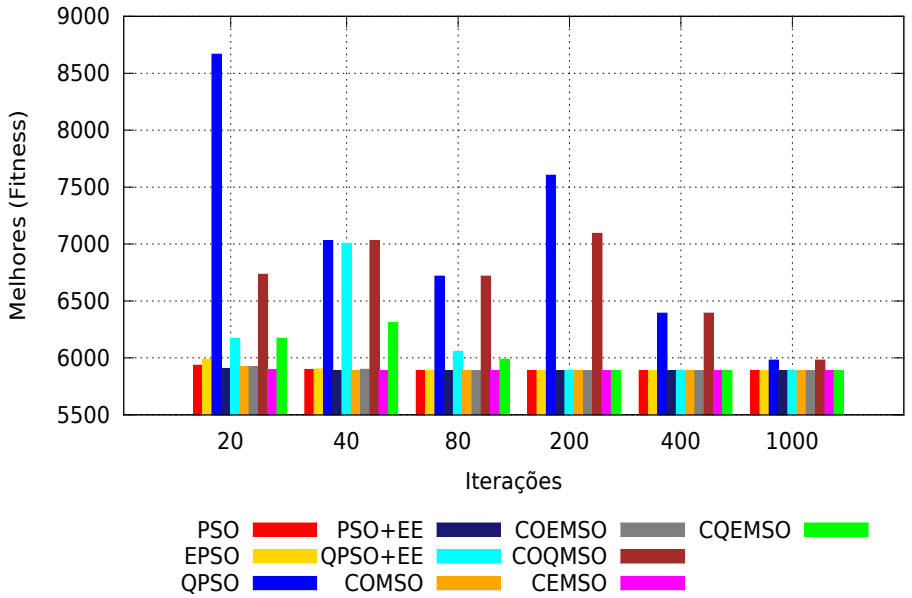
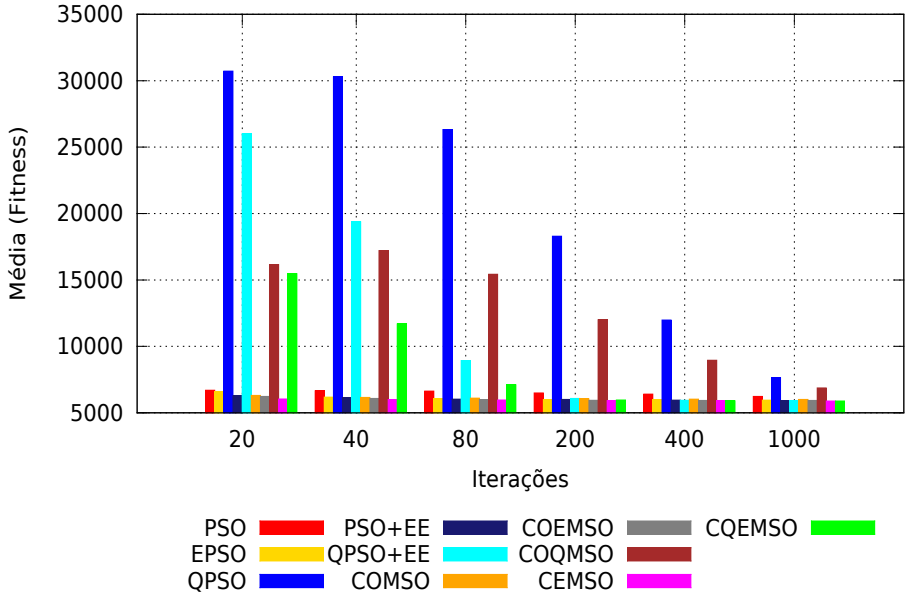


Figura 4.20: Gráfico comparativo das médias para o DPV



As Figuras 4.21 e 4.22 mostram, respectivamente, a varância dos resultados encontrados e o tempo de execução de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.21: Gráfico comparativo dos valores de variância para o DPV

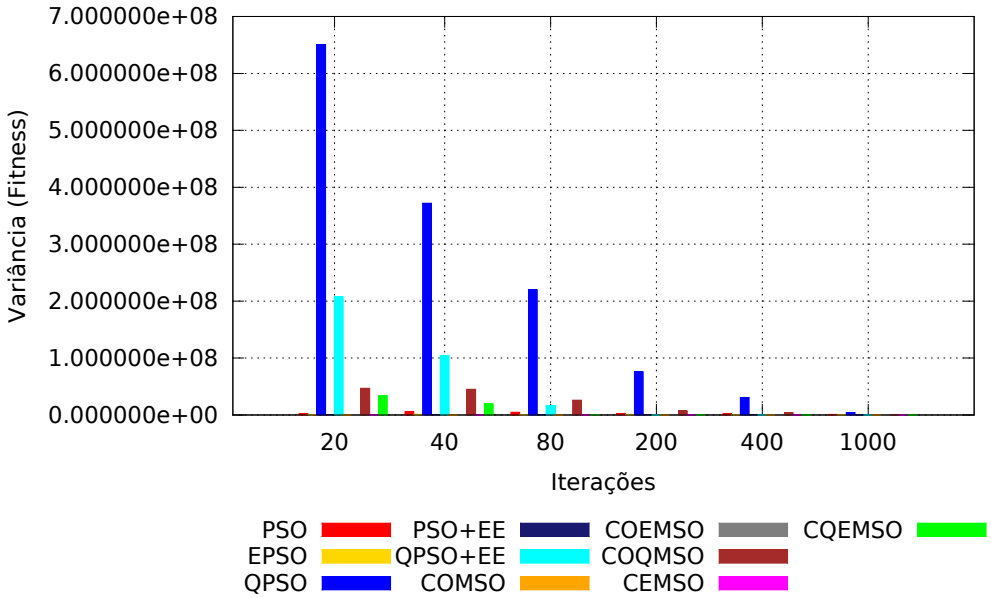
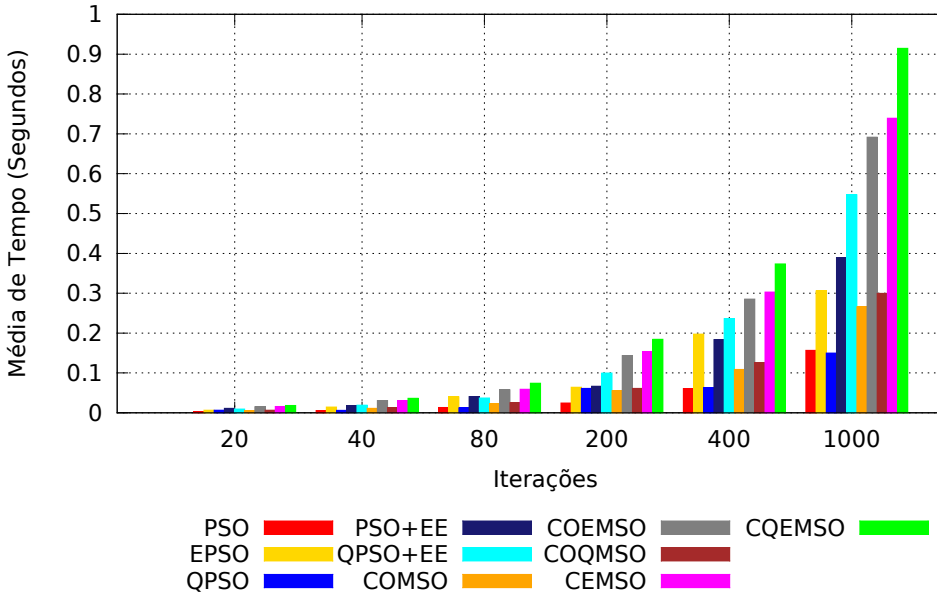


Figura 4.22: Gráfico comparativo da média do tempo de execução dos algoritmos testados para o DPV



As Figuras 4.23 e 4.24 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.23: Convergência dos enxames escravos do algoritmo COMSO para o DPV

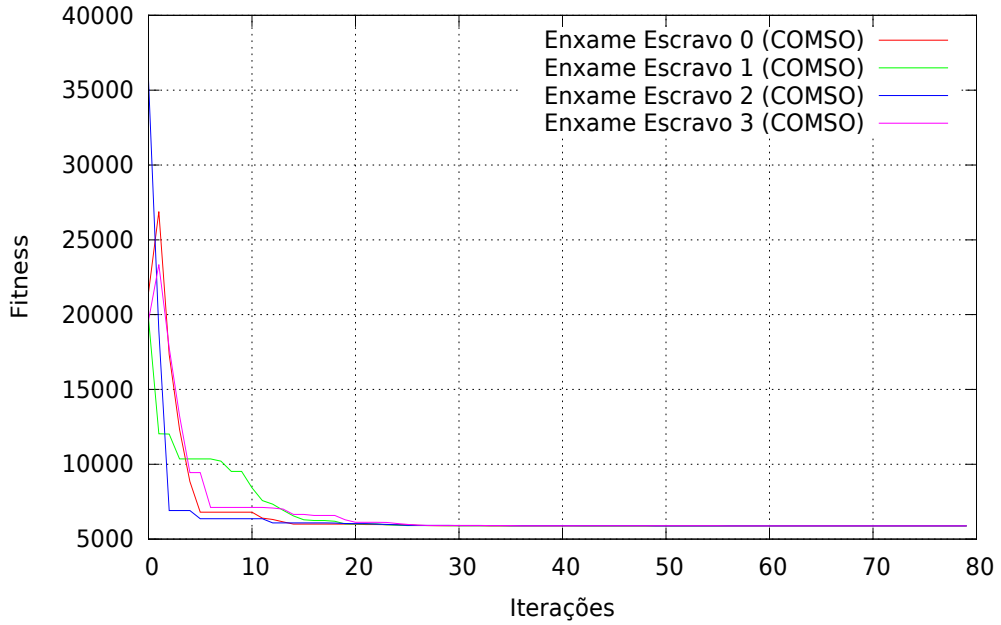
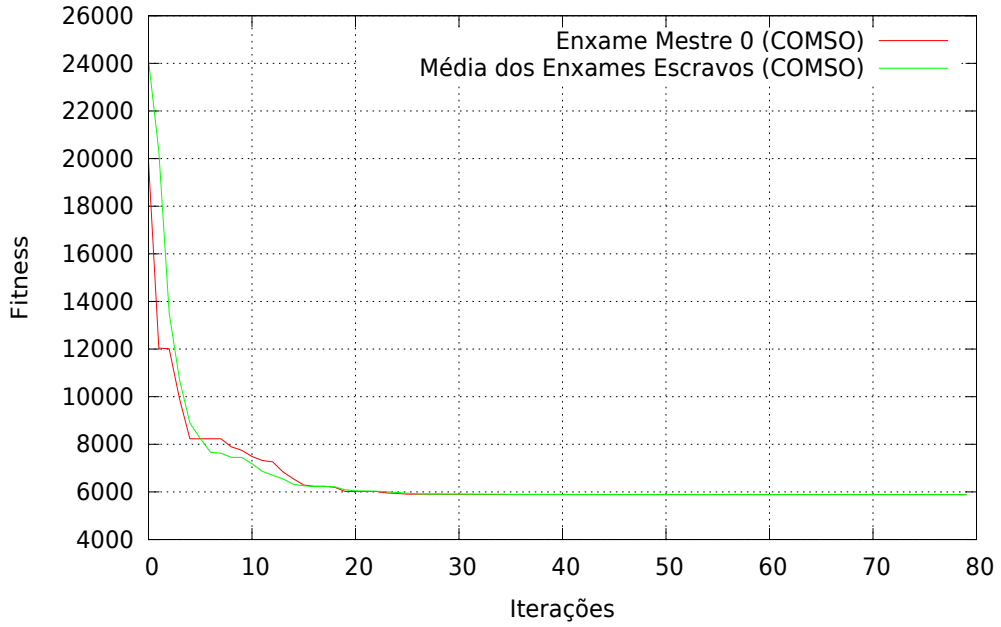


Figura 4.24: Convergência do enxame mestre do algoritmo COMSO para o DPV



As Figuras 4.25 e 4.26 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.25: Convergência dos enxames escravos do algoritmo COEMSO para o DPV

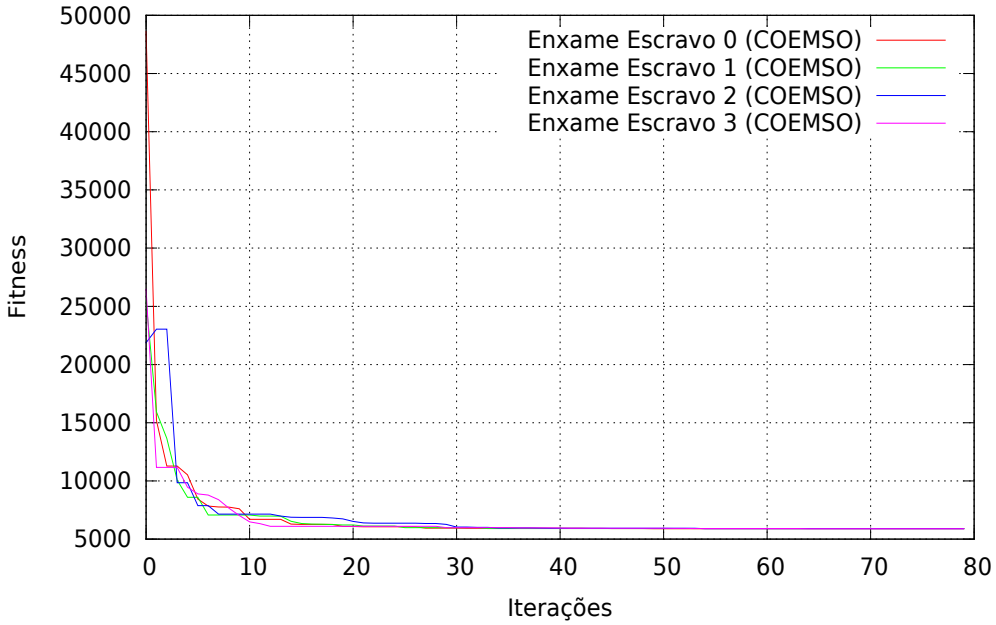
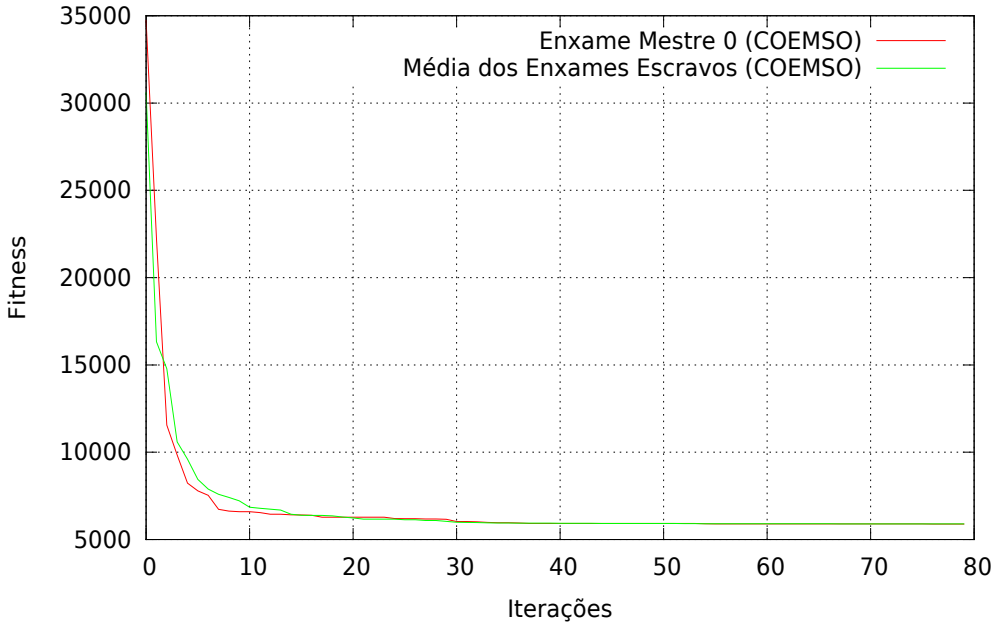


Figura 4.26: Convergência do enxame mestre do algoritmo COEMSO para o DPV



As Figuras 4.27 e 4.28 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COQMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.27: Convergência dos enxames escravos do algoritmo COQMSO para o DPV

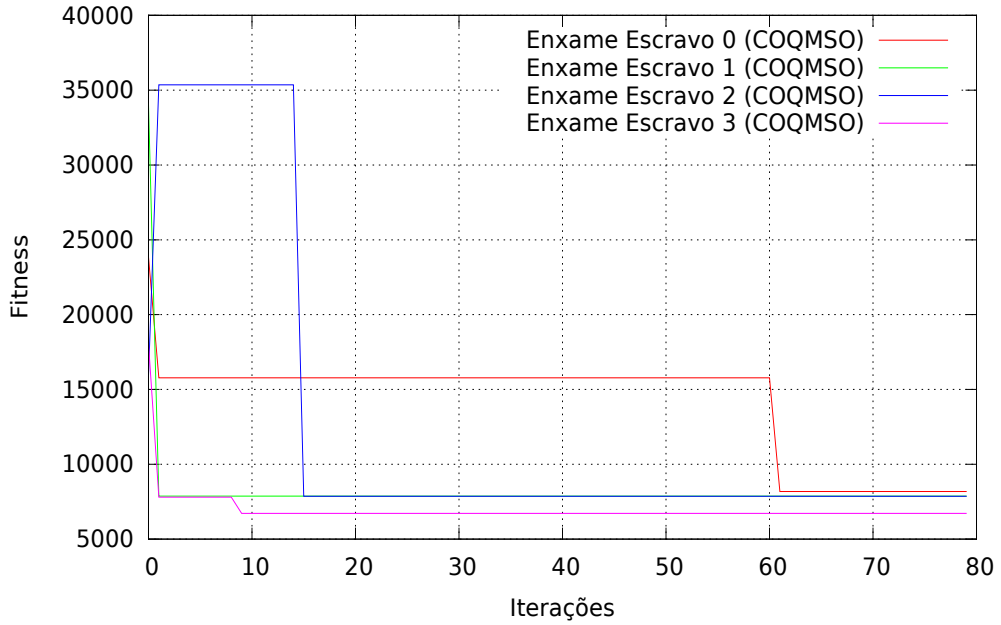
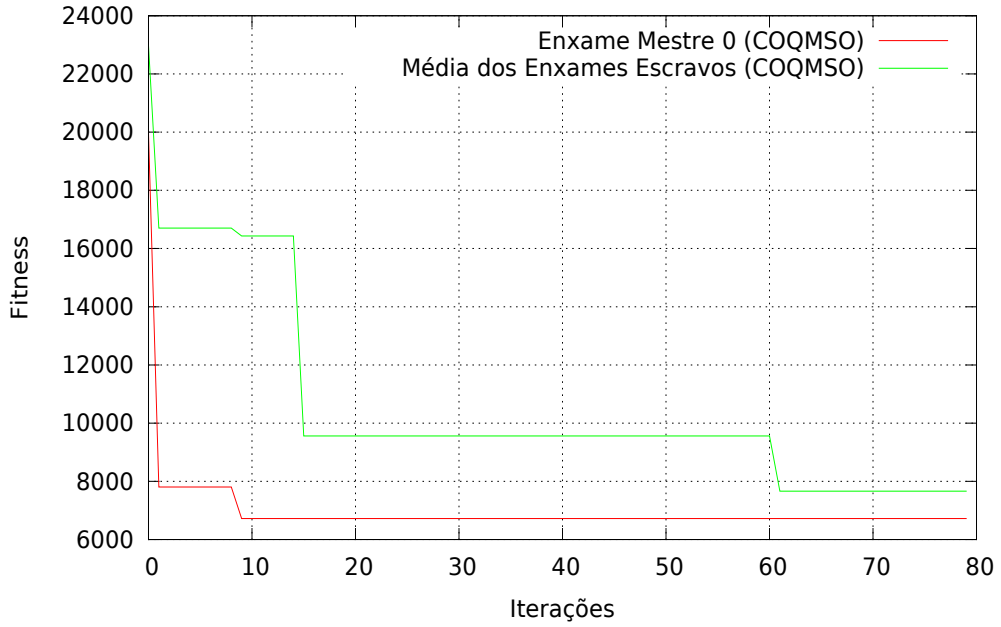


Figura 4.28: Convergência do enxame mestre do algoritmo COQMSO para o DPV



As Figuras 4.29 e 4.30 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.29: Convergência dos enxames escravos do algoritmo CEMSO para o DPV

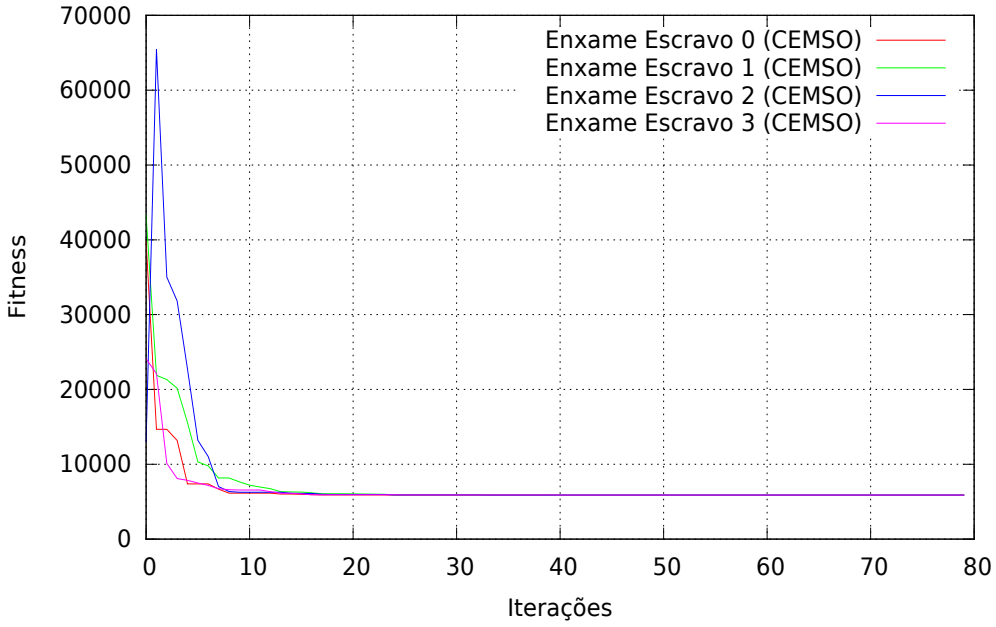
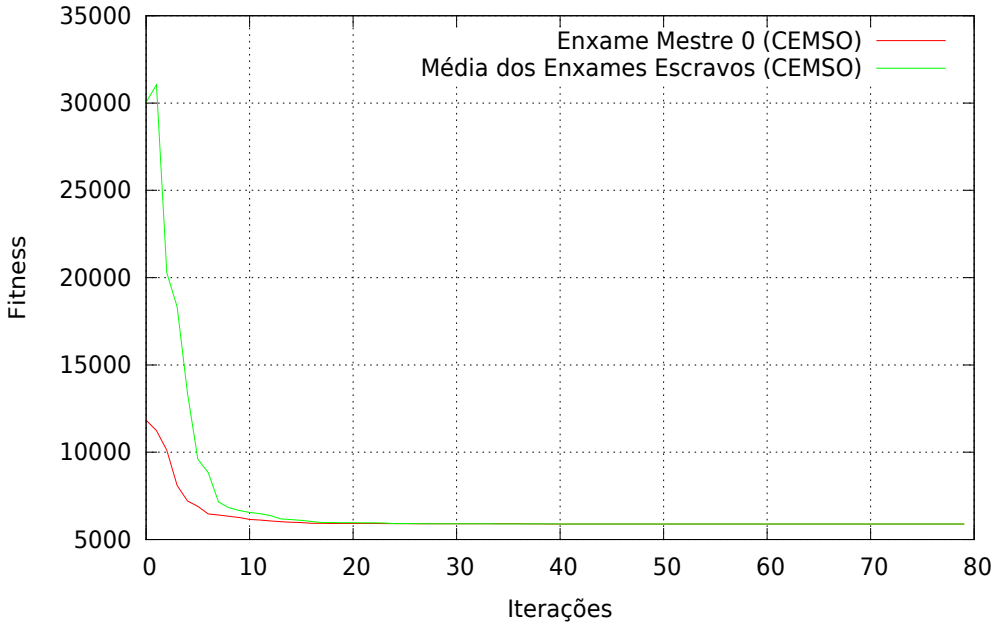


Figura 4.30: Convergência do enxame mestre do algoritmo CEMSO para o DPV



As Figuras 4.31 e 4.32 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.31: Convergência dos enxames escravos do algoritmo CQEMSO para o DPV

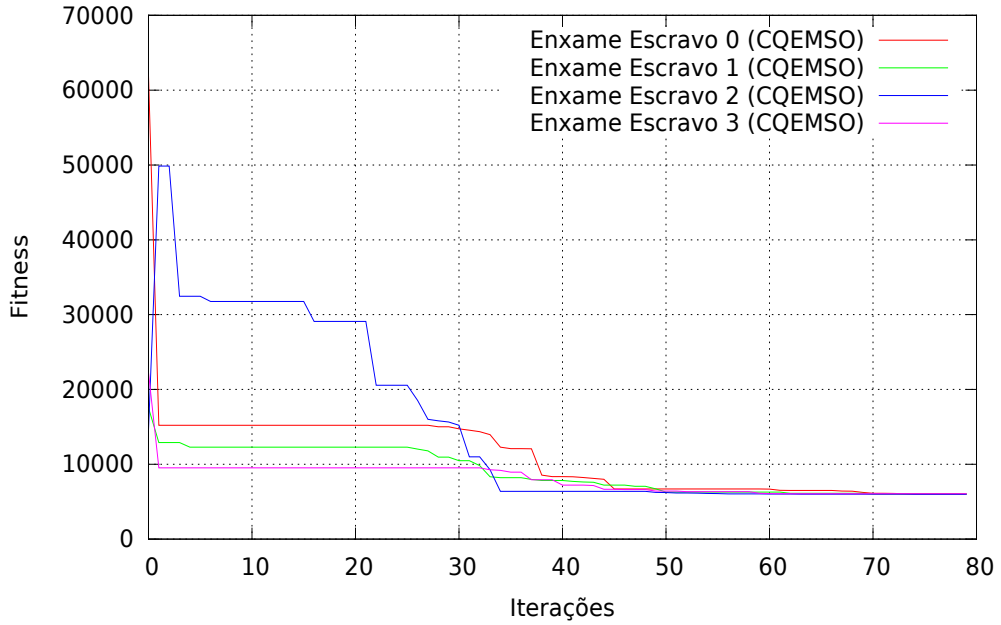
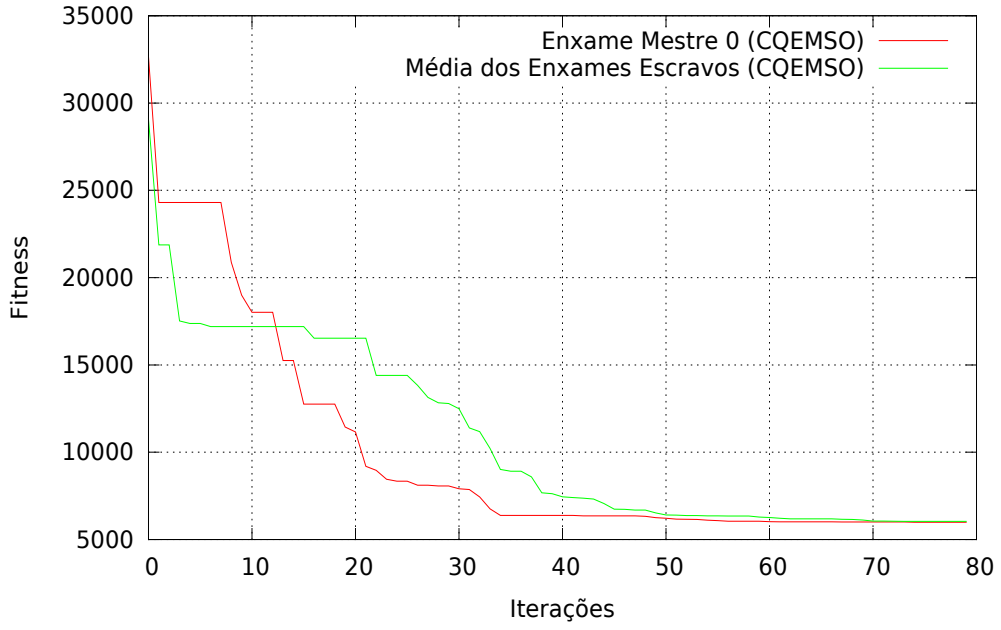


Figura 4.32: Convergência do enxame mestre do algoritmo CQEMSO para o DPV



As Figuras 4.33 e 4.34 mostram a média de convergência dos enxames escravos dos algoritmos COMSO, COEMSO, CEMSO, COQMSO e CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.33: Média de convergência dos enxames escravos para os algoritmo COMSO, CEMSO e COEMSO no problema DPV

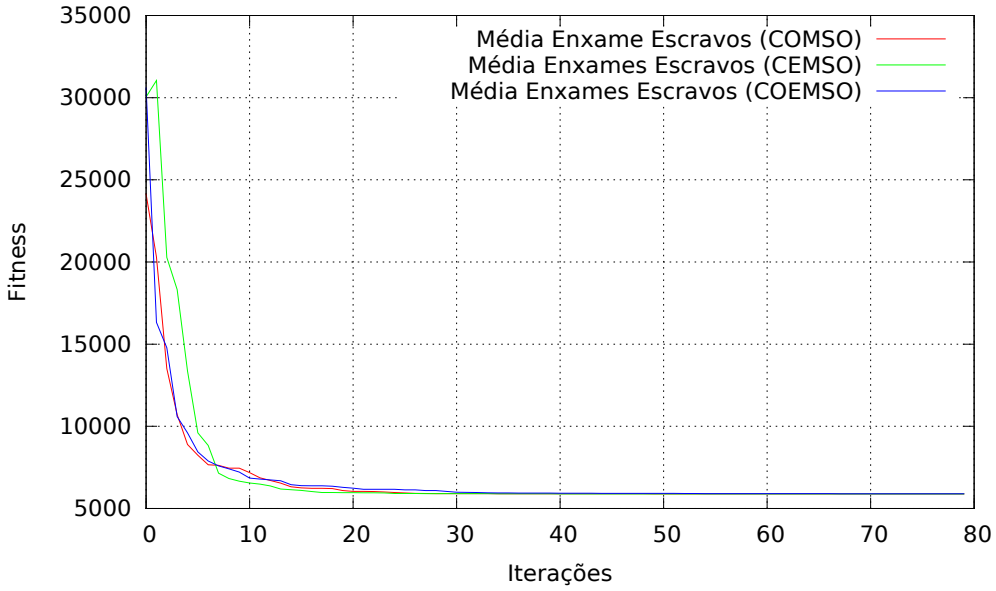
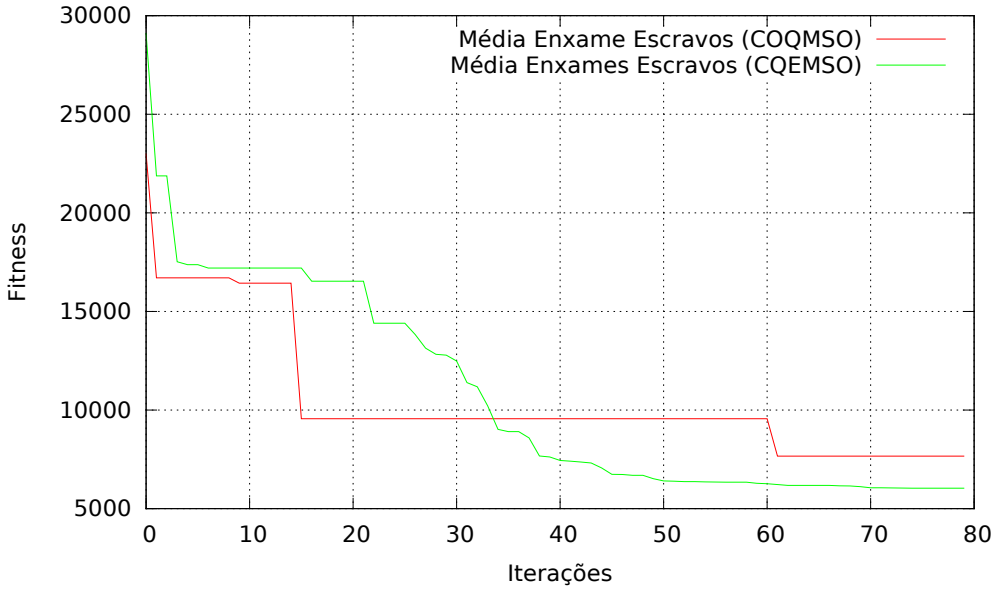


Figura 4.34: Média de convergência dos enxames escravos para os algoritmo COQMSO e CQEMSO no problema DPV



Análise dos Resultados para o DPV

Com os resultados obtidos pelos algoritmos CEMSO e CQEMSO, é possível estabelecer uma comparação dos resultados obtidos nessas simulações com resultados presentes na literatura. São utilizados os resultados do melhor *fitness* obtido através do experimento de 1000 iterações.

Segue-se os algoritmos usados para comparação:

- ***Alg. 1 [Teixeira 2012]***: Algoritmo genético com interação social baseado na teoria dos jogos (SIGA);
- ***Alg. 2 [Coello & Montes 2002]***: Algoritmo genético com mecanismo de manipulação de restrição.
- ***Alg. 3 [Cagnina et al. 2008]***: Otimização por enxame de partículas adapatado para problemas de engenharia com equação de velocidade modificada.

Tabela 4.14: Comparação dos resultados para o problema DPV

Variaveis	CEMSO	CQEMSO	Alg. 1	Alg. 2	Alg. 3
T_s	0.778169	0.778169	0.812500	0.812500	0.812500
T_h	0.384649	0.384649	0.437500	0.437500	0.437500
R	40.319622	40.319626	42.092732	42.097398	42.098445
L	200.000000	199.999954	195.678619	176.654050	176.636595
G_1	0.000000	-0.000000	-0.000110	-0.000020	-4.500e-15
G_2	0.000000	-0.000000	-0.035935	-0.035891	-0.035880
G_3	0.000000	-0.000000	-1337.994634	-27.886075	-1.164e-10
G_4	-40.000000	-40.000046	-63.052220	-63.345953	-63.363404
Violações	0	0	0	0	0
Tempo	0.728123	0.945307	N/A	N/A	N/A
Fitness	5885.333984	5885.334961	6066.029360	6059.946300	6059.714335

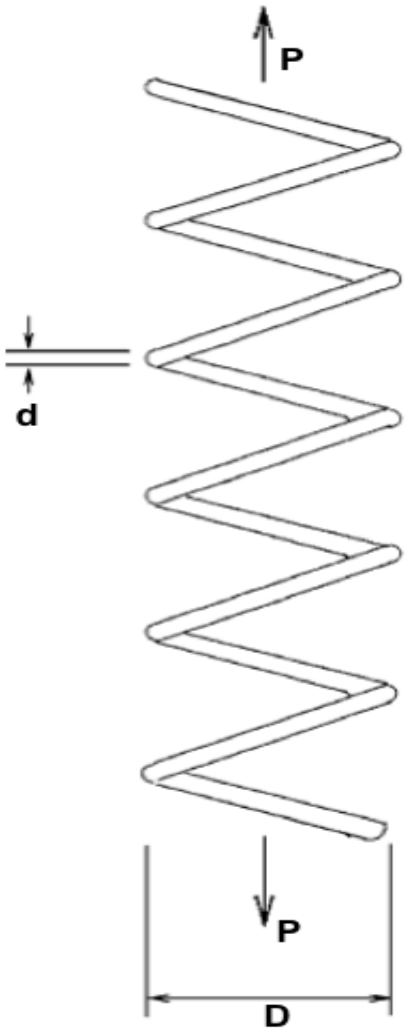
- O comportamento de convergência entre a média dos enxames escravos e enxame mestre indicam os seguintes resultados:
    1. O algoritmo COMSO demonstra uma convergência acentuada dos enxames escravos, com resultados superiores em relação ao enxame mestre, sendo superado a partir da iteração 25 (Figura 4.24);
    2. O algoritmo COEMSO demonstra uma convergência acentuada dos enxames escravos, com resultados superiores em relação ao enxame mestre, sendo superado a partir da iteração 33 (Figura 4.26);
    3. O algoritmo CQEMSO apresentou uma convergência gradual dos enxames escravos, com resultados superiores em relação ao enxame mestre, sendo superado a partir da iteração 12 (Figura 4.32);
- O algoritmo CEMSO apresentou os melhores resultados em termos de média, variância e melhores resultados em todos os experimentos;
- O comportamento de convergência observado no algoritmo CQEMSO (Gráficos 4.31 e 4.32) demonstram uma variabilidade maior no processo de busca entre as iterações;
- Apesar do desempenho obtido pelos algoritmos COMSO, COEMSO e COQMSO, os algoritmos CEMSO e CQEMSO apresentaram uma média de convergência superior dentre as soluções comparadas;
- O melhor resultado encontrado pelos algoritmos com base no PSO para as partículas originais foi 5885.333984 e foram obtidos pelos seguintes algoritmos:
    1. CEMSO: Obteve o melhor resultado a partir de 400 iterações;
    2. COMSO: Obteve o melhor resultado a partir de 400 iterações;
    3. PSO: Obteve o melhor resultado a partir de 1000 iterações;
    4. PSO+EE: Obteve o melhor resultado a partir de 1000 iterações.
- O melhor resultado encontrado pelos algoritmos com base no EPSO para as partículas originais foi 5885.336914 e foram obtidos pelos seguintes algoritmos:
    1. COEMSO: Obteve o melhor resultado a partir de 1000 iterações;
    2. EPSO: Obteve o melhor resultado a partir de 1000 iterações.
- O melhor resultado encontrado pelos algoritmos com base no QPSO para as partículas originais foi 5885.334961 e foram obtidos pelos seguintes algoritmos:
    1. CQEMSO: Obteve o melhor resultado a partir de 1000 iterações;
    2. QPSO: Melhor resultado obtido for 5885.334961 (1000 iteração);
    3. QPSO+EE: Obteve o melhor resultado a partir de 1000 iterações.

- Em alguns experimentos, os algoritmos multi-enxame apresentaram o melhor resultado encontrado pelos subenxames como a melhor solução: COMSO (40 e 1000 iterações), COEMSO (400 e 1000 iterações), COQMSO (40, 80, 400 e 1000 iterações), CEMSO (1000 iterações) e CQEMSO (20 iterações);
- Durante os experimentos realizados, pode-se observar que a diferença de tempo entre os algoritmos que utilizam a topologia multi-enxame é relativamente pequena. Por outro lado, os algoritmos que utilizam os mecanismos de estratégias evolutivas (EPSO, COEMSO, CEMSO e CQEMSO) apresentaram um tempo de execução maior;
- Ao serem comparados com os resultados de outras soluções encontradas na literatura (expostos na Tabela 4.14), os algoritmos CEMSO e CQEMSO obtiveram os melhores resultados.

4.1.3 Peso da Tensão/Compressão sobre Mola

Segundo [Coello & Montes 2002], o esquema de Peso da Tensão/Compressão sobre Mola (MWTCS) consiste na minimização do peso da tensão/compressão sobre uma mola, que está sujeita a algumas restrições, tais como: deflexão mínima, tensão do cisalhamento, a frequência de onda, limites de diâmetro externo e variaáveis de projeto. As variáveis do problema MWTCS são: Diâmetro do fio (d), bobina de diâmetro médio (D) e o número de bobinas ativas (P). A Figura 4.35 mostra o esquema do MWTCS.

Figura 4.35: Esquema do *Minimization of Weight of Tension/Compression String* (MWTCS). Fonte:[Coello & Montes 2002]



O MWTCS pode ser formalizado matematicamente através do seguinte esquema:

$$\text{Minimizar } f(X_i) = (P+2)Dd^2 \tag{4.27}$$

Sujeito a:

$$g_1 = 1 - \frac{D^3 P}{71785d^4} \leq 0 \tag{4.28}$$

$$g_2 = \frac{4D^2 - dD}{12566(Dd^3 - d^4)} + \frac{1}{5108d^2} - 1 \leq 0 \tag{4.29}$$

$$g_3 = 1 - \frac{140.45d}{D^2 P} \leq 0 \tag{4.30}$$

$$g_4 = \frac{d-D}{1.5} - 1 \leq 0 \tag{4.31}$$

Os limites das variáveis d, D e P são: $0.05 \leq d \leq 2.0$; $0.25 \leq D \leq 1.3$; $2.0 \leq P \leq 15.0$.

No intuito de avaliar os algoritmos CEMSO e CQEMSO, foram utilizadas as seguintes configurações: Iterações = 4, 8, 10, 20, 40 e 80; quantidade de execuções = 500; quantidade de partículas para cada enxame escravo = 80; quantidade de réplicas por partícula = 4; quantidade de enxames escravos = 4; constante de pertubação para o melhor global (σ_g) = 0.005; parâmetro de estratégias para a mutação dos fatores de inércia e de aceleração (σ) = 0.22; fatores C_1 e C_2 (COEMSO e CEMSO) = 2.05; fator de aceleração para o enxame mestre (C_3) = 2.02; quantidade de testes executados para cada algoritmo = 500; Fator de diminuição de velocidade (δ) = diminuir a velocidade a 40% (0.4); Fator de penalidade para resultados com violações = 200; Constante de probabilidade (θ) = 0.5; Velocidade mínima (V_{MIN}) = metade do valor de limite máximo de posição com sinal ou direção invertida ($-(\frac{X_{MAX}}{2})$). Os valores de fator de inércia (w) (CEMSO), bem como fatores α e β (CQEMSO e COQMSO) são gerados a cada iteração pelas Equações 2.2, 2.30 e 2.32 (decrescimento randômico).

Além dos algoritmos CEMSO e CQEMSO, foram utilizados para efeito de comparação os algoritmos PSO, EPSO, QPSO, PSO+EE e QPSO+EE baseados na implementação exposta nos capítulos 3 e 4, bem como suas versões multi-enxame (COMSO, COEMSO e COQMSO). As Tabelas 4.15, 4.16, 4.17, 4.18, 4.19 e 4.20 mostram os resultados obtidos nos experimentos realizados para o problema MWTCS.

Tabela 4.15: Comparação dos resultados para 4 iterações para o MWTCS

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	0.0038354	0.002822	0	$1.147776e-05$	0.000938 s
EPSO	0.0031470	0.002821	0	$7.336633e-08$	0.001729 s
QPSO	0.0097550	0.002834	0	$8.916715e-05$	0.000931 s
PSO+EE	0.0029752	0.002820	0	$3.136138e-08$	0.001671 s
QPSO+EE	0.0032328	0.002820	0	$2.644268e-07$	0.001936 s
COMSO	0.0028872	0.002820	0	$5.808943e-09$	0.001381 s
COEMSO	0.0028653	0.002820	0	$1.789142e-09$	0.002595 s
COQMSO	0.0033680	0.002822	0	$3.022101e-07$	0.001284 s
CEMSO	0.0028430	0.002820	0	$5.822316e-10$	0.002947 s
CQEMSO	0.0028729	0.002820	0	$2.682292e-09$	0.003119 s

Tabela 4.16: Comparação dos resultados para 8 iterações para o MWTCS

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	0.0028657	0.002820	0	$8.758780e-09$	0.001283 s
EPSO	0.0028722	0.002820	0	$2.845904e-09$	0.003771 s
QPSO	0.0031827	0.002821	0	$4.659803e-07$	0.001673 s
PSO+EE	0.0028295	0.002820	0	$2.416778e-10$	0.003555 s
QPSO+EE	0.0028678	0.002820	0	$5.133171e-09$	0.002898 s
COMSO	0.0028243	0.002820	0	$1.688994e-11$	0.002167 s
COEMSO	0.0028247	0.002820	0	$2.794385e-11$	0.005914 s
COQMSO	0.0028659	0.002820	0	$2.182373e-09$	0.002698 s
CEMSO	0.0028211	0.002820	0	$9.447240e-13$	0.005307 s
CQEMSO	0.0028254	0.002820	0	$4.323487e-11$	0.006609 s

Tabela 4.17: Comparação dos resultados para 10 iterações para o MWTCS

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	0.0028362	0.002820	0	$8.711342e-10$	0.001188 s
EPSO	0.0028628	0.002820	0	$2.447299e-09$	0.005393 s
QPSO	0.0029703	0.002820	0	$3.268651e-08$	0.001275 s
PSO+EE	0.0028234	0.002820	0	$3.928844e-11$	0.003630 s
QPSO+EE	0.0028433	0.002820	0	$1.792909e-09$	0.004381 s
COMSO	0.0028219	0.002820	0	$3.217771e-12$	0.003105 s
COEMSO	0.0028236	0.002821	0	$2.682812e-11$	0.007554 s
COQMSO	0.0028405	0.002820	0	$4.636143e-10$	0.002966 s
CEMSO	0.0028204	0.002820	0	$5.853120e-14$	0.007066 s
CQEMSO	0.0028226	0.002820	0	$1.068471e-11$	0.008309 s

Tabela 4.18: Comparação dos resultados para 20 iterações para o MWTCS

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	0.0028742	0.002820	0	$5.540590e-08$	0.003355 s
EPSO	0.0028281	0.002820	0	$2.647469e-10$	0.009057 s
QPSO	0.0028366	0.002820	0	$6.760482e-10$	0.003935 s
PSO+EE	0.0028203	0.002820	0	$6.414367e-13$	0.007268 s
QPSO+EE	0.0028219	0.002820	0	$7.769329e-12$	0.009943 s
COMSO	0.0028210	0.002820	0	$5.413947e-11$	0.005435 s
COEMSO	0.0028207	0.002820	0	$3.463857e-13$	0.014432 s
COQMSO	0.0028228	0.002820	0	$9.851167e-12$	0.006405 s
CEMSO	0.0028202	0.002820	0	$1.437386e-17$	0.014147 s
CQEMSO	0.0028205	0.002820	0	$8.528664e-14$	0.016434 s

Tabela 4.19: Comparação dos resultados para 40 iterações para o MWTCS

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	0.0028433	0.002820	0	$2.597027e-08$	0.006456 s
EPSO	0.0028214	0.002820	0	$1.218846e-11$	0.013083 s
QPSO	0.0028241	0.002820	0	$3.437550e-11$	0.006922 s
PSO+EE	0.0028202	0.002820	0	$1.194322e-14$	0.017122 s
QPSO+EE	0.0028204	0.002820	0	$6.607992e-14$	0.017731 s
COMSO	0.0028204	0.002820	0	$6.347296e-12$	0.011330 s
COEMSO	0.0028203	0.002820	0	$6.210281e-15$	0.028532 s
COQMSO	0.0028208	0.002820	0	$5.144376e-13$	0.011695 s
CEMSO	0.0028202	0.002820	0	$5.885197e-20$	0.029645 s
CQEMSO	0.0028203	0.002820	0	$1.429760e-15$	0.032760 s

Tabela 4.20: Comparação dos resultados para 80 iterações para o MWTCS

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	0.0028468	0.002820	0	$4.948406e-08$	0.010299 s
EPSO	0.0028207	0.002820	0	$8.151207e-12$	0.028823 s
QPSO	0.0028210	0.002820	0	$9.607521e-13$	0.010943 s
PSO+EE	0.0028202	0.002820	0	$1.847869e-16$	0.024557 s
QPSO+EE	0.0028202	0.002820	0	$1.070633e-15$	0.038571 s
COMSO	0.0028203	0.002820	0	$2.369036e-12$	0.021794 s
COEMSO	0.0028202	0.002820	0	$1.679205e-16$	0.057139 s
COQMSO	0.0028204	0.002820	0	$2.717483e-14$	0.023303 s
CEMSO	0.0028200	0.002820	0	$0.000000e+00$	0.056106 s
CQEMSO	0.0028202	0.002820	0	$1.389085e-17$	0.065769 s

As Figuras 4.36 e 4.37 mostram, respectivamente, o comparativo entre os melhores valores obtidos e a média dos resultados encontrados de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.36: Gráfico comparativo dos melhores resutados obtidos para o MWTCS

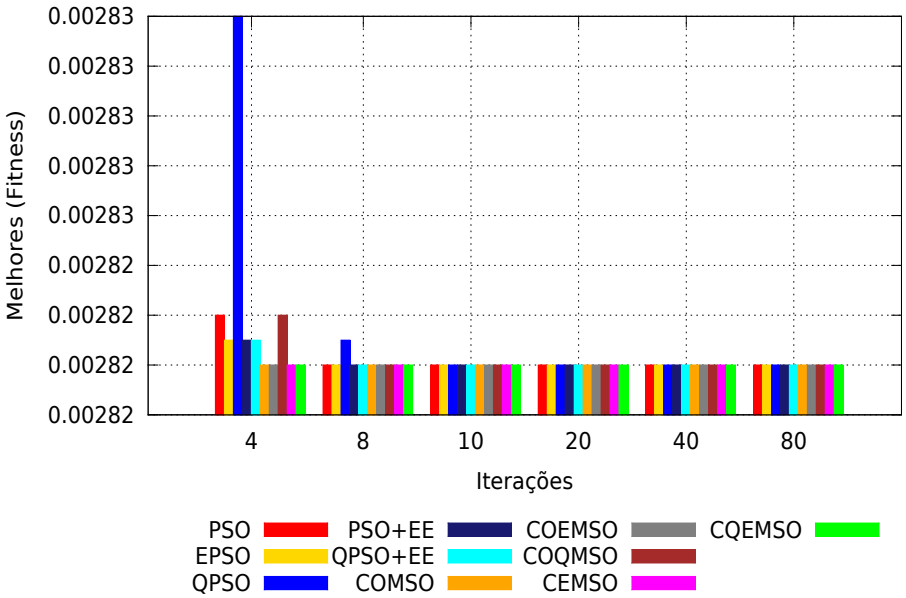
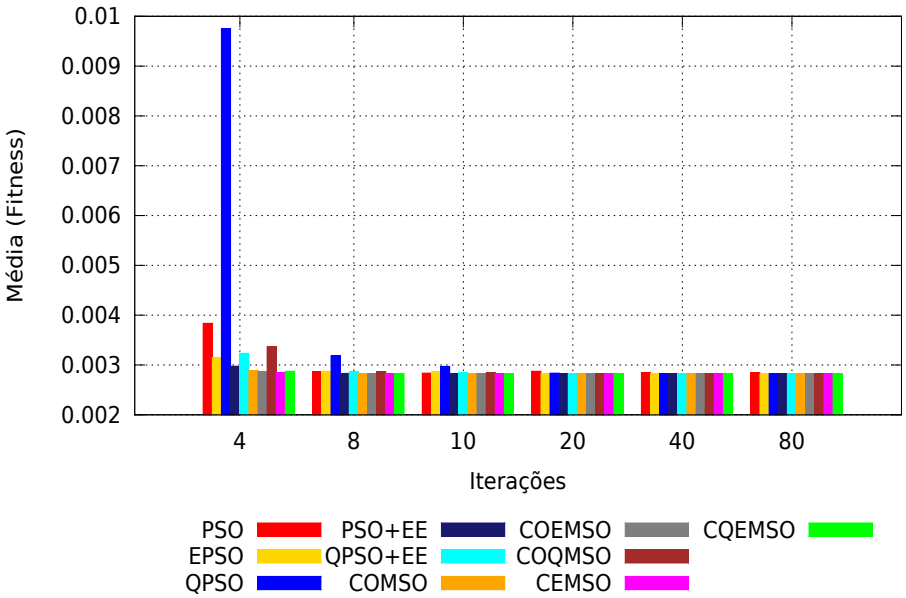


Figura 4.37: Gráfico comparativo das médias para o MWTCS



As Figuras 4.38 e 4.39 mostram, respectivamente, a varância dos resultados encontrados e o tempo de execução de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.38: Gráfico comparativo dos valores de variância para o MWTCS

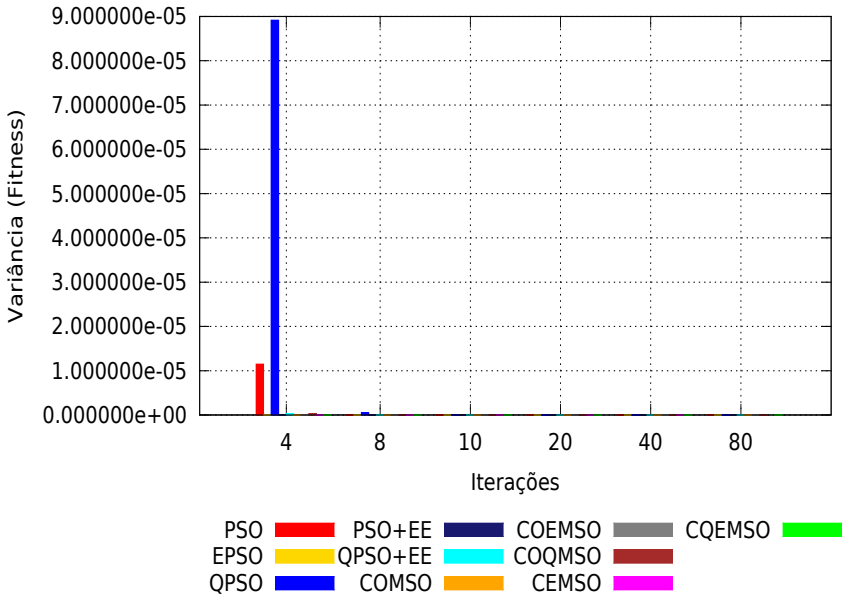
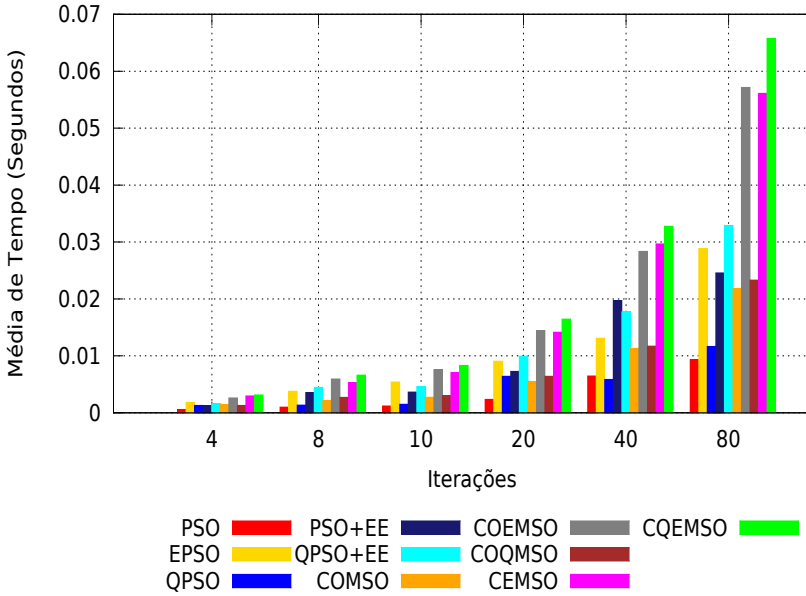


Figura 4.39: Gráfico comparativo da média do tempo de execução dos algoritmos testados para o MWTCS



As Figuras 4.40 e 4.41 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COMSO, obtidos nos melhores resultados encontrados para o experimento de 20 iterações.

Figura 4.40: Convergência dos enxames escravos do algoritmo COMSO para o MWTCS

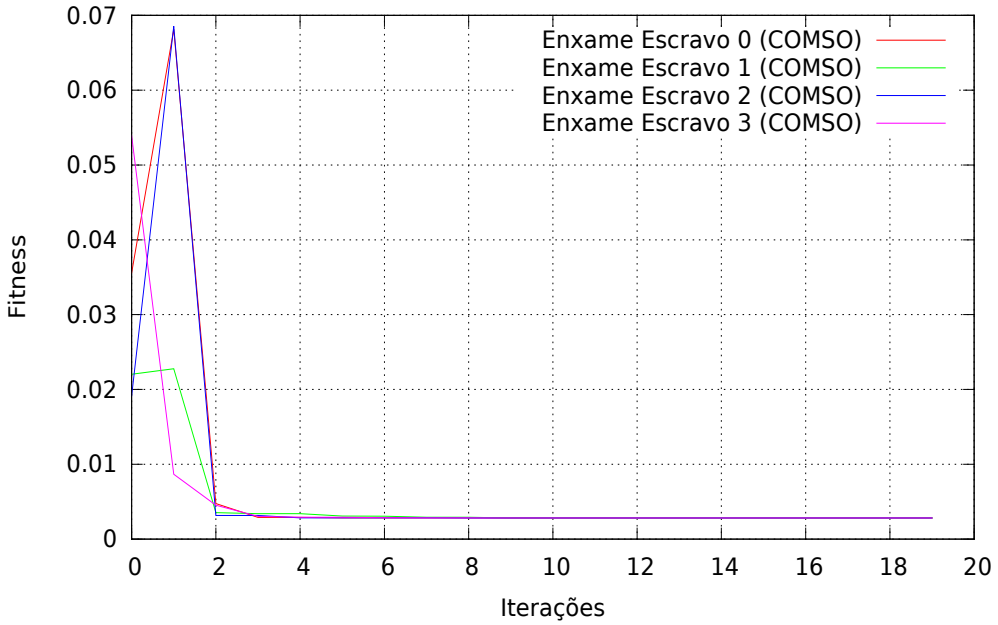
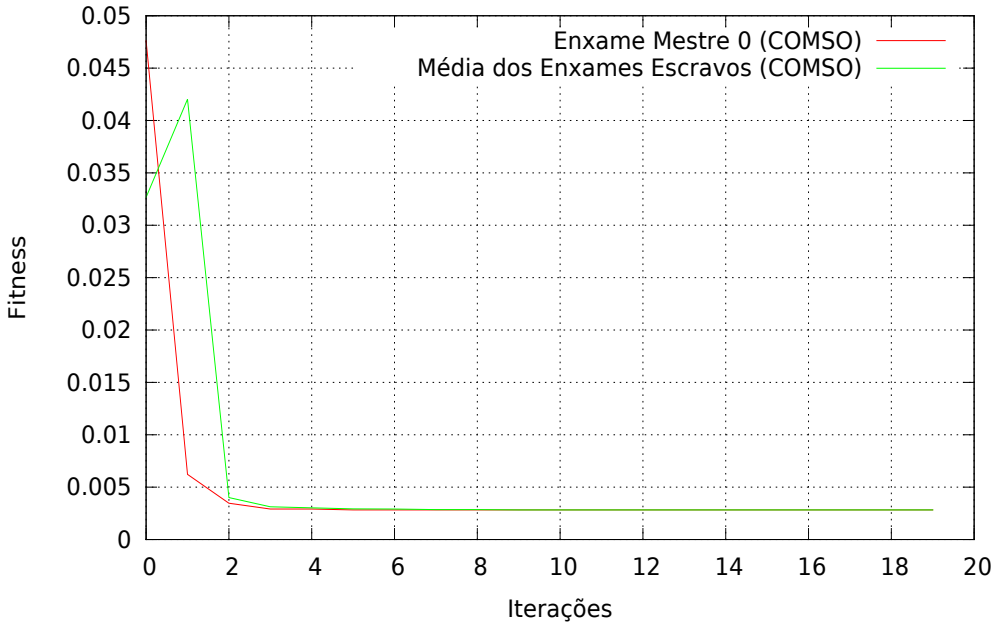


Figura 4.41: Convergência do enxame mestre do algoritmo COMSO para o MWTCS



As Figuras 4.42 e 4.43 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COEMSO, obtidos nos melhores resultados encontrados para o experimento de 20 iterações.

Figura 4.42: Convergência dos enxames escravos do algoritmo COEMSO para o MWTCS

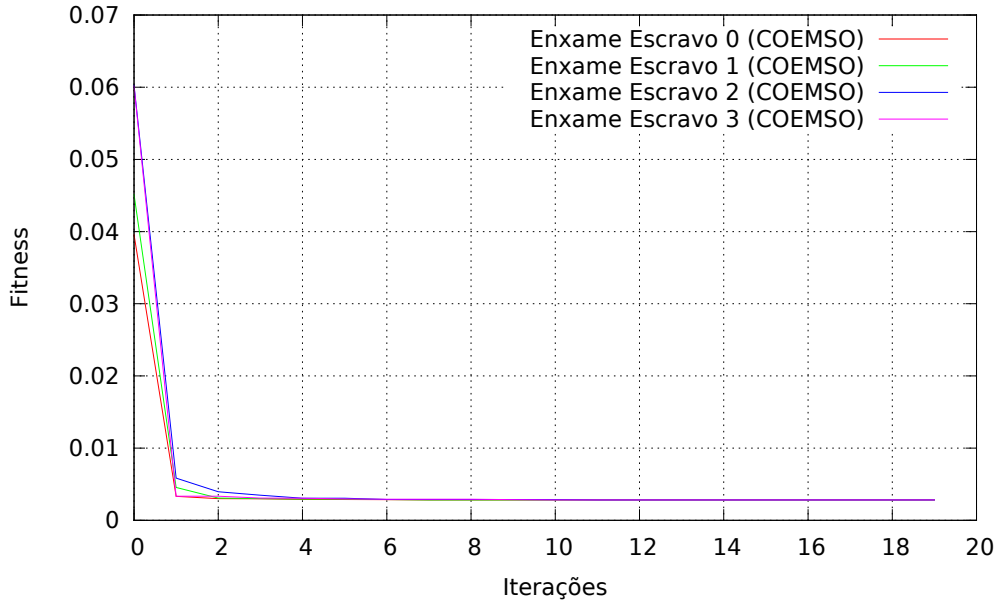
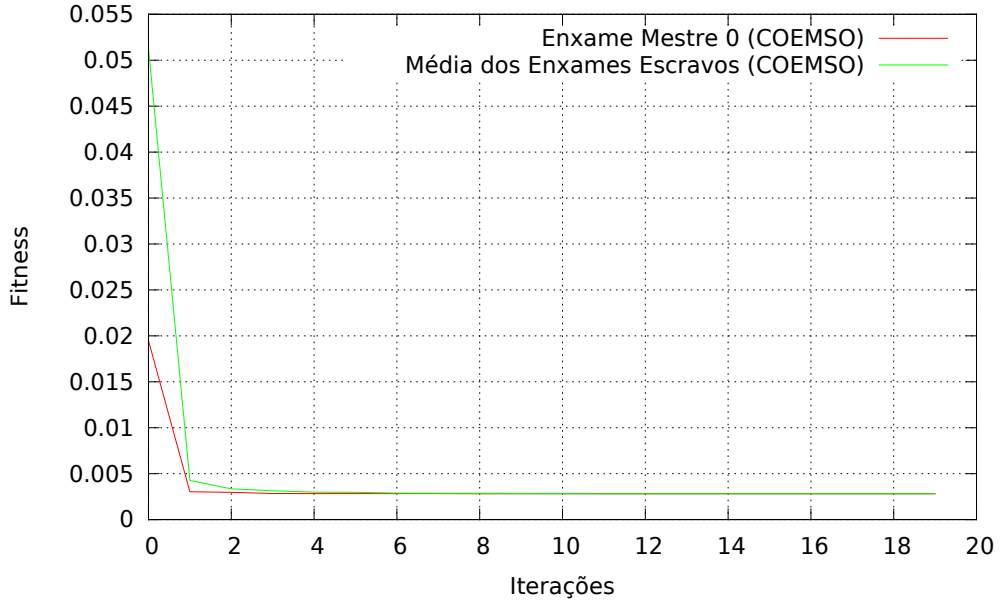


Figura 4.43: Convergência do enxame mestre do algoritmo COEMSO para o MWTCS



As Figuras 4.44 e 4.45 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COQMSO, obtidos nos melhores resultados encontrados para o experimento de 20 iterações.

Figura 4.44: Convergência dos enxames escravos do algoritmo COQMSO para o MWTCS

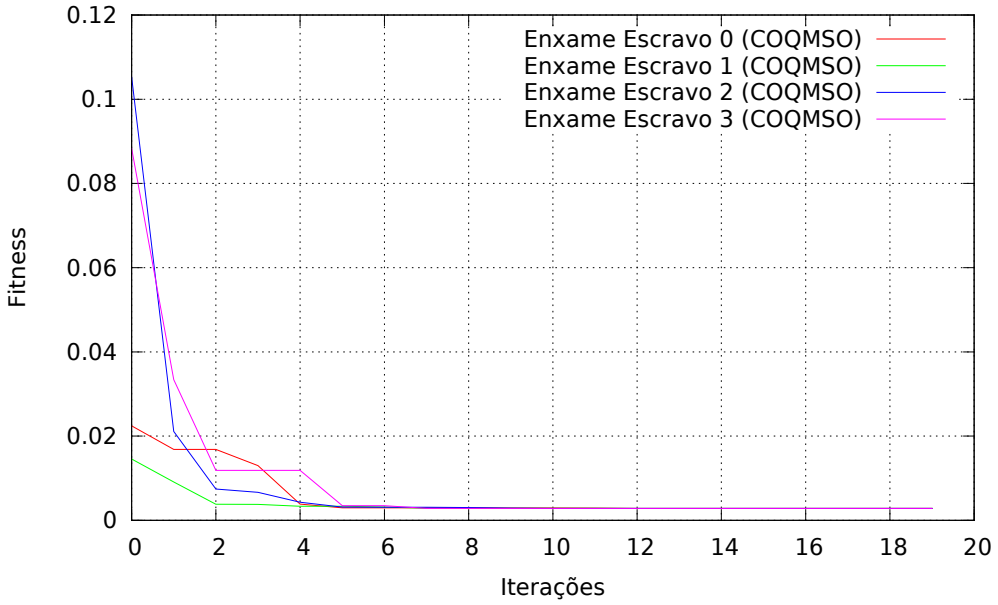
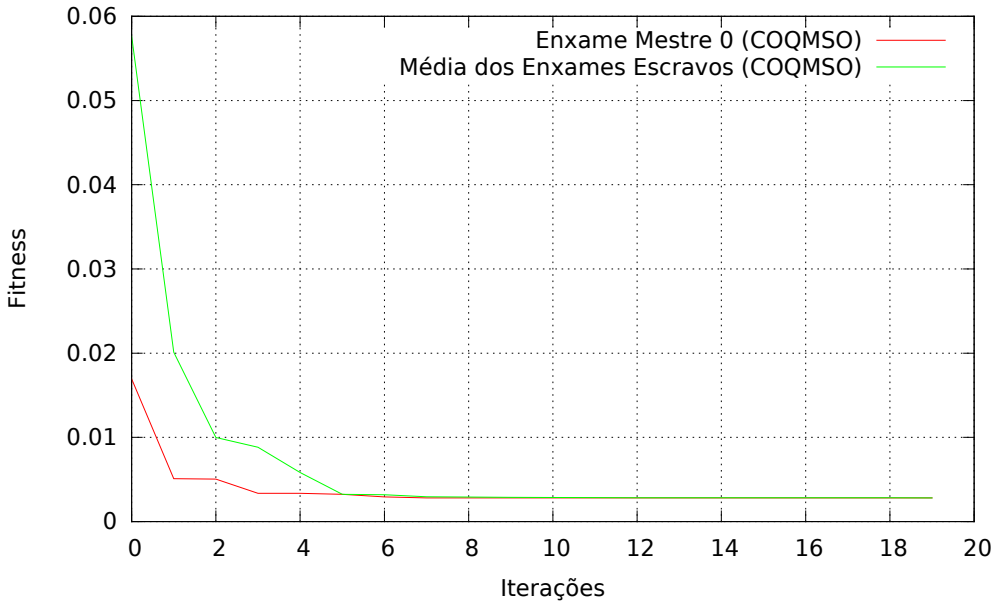


Figura 4.45: Convergência do enxame mestre do algoritmo COQMSO para o MWTCS



As Figuras 4.46 e 4.47 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CEMSO, obtidos nos melhores resultados encontrados para o experimento de 20 iterações.

Figura 4.46: Convergência dos enxames escravos do algoritmo CEMSO para o MWTCS

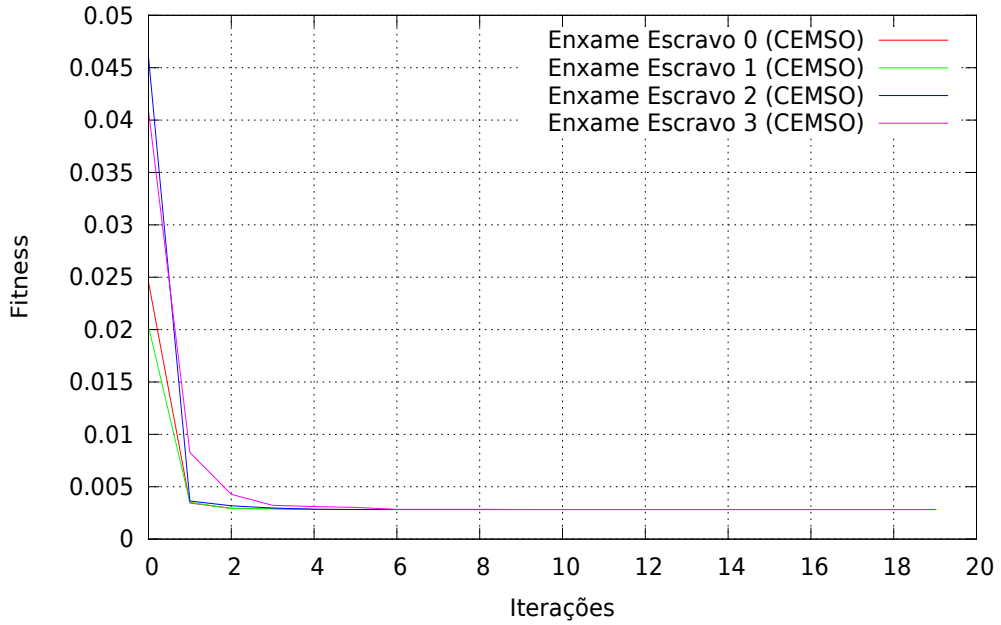
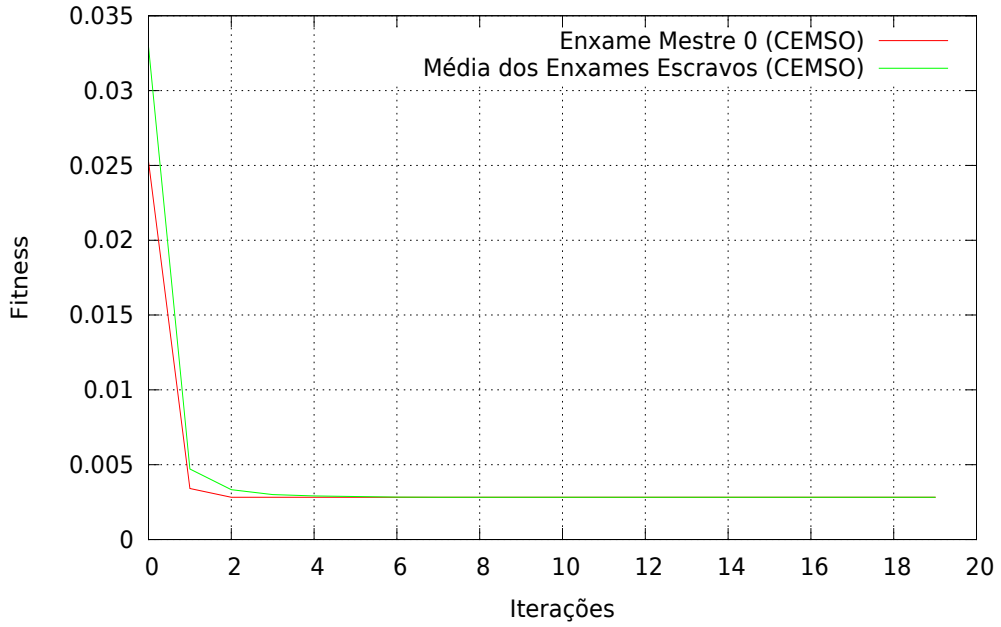


Figura 4.47: Convergência do enxame mestre do algoritmo CEMSO para o MWTCS



As Figuras 4.48 e 4.49 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 20 iterações.

Figura 4.48: Convergência dos enxames escravos do algoritmo CQEMSO para o MWTCS

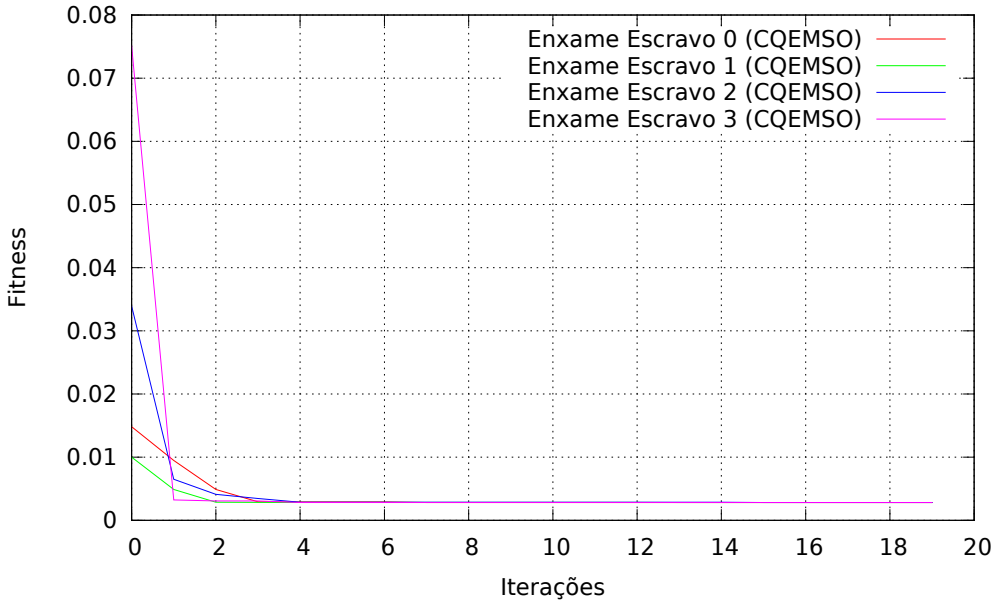
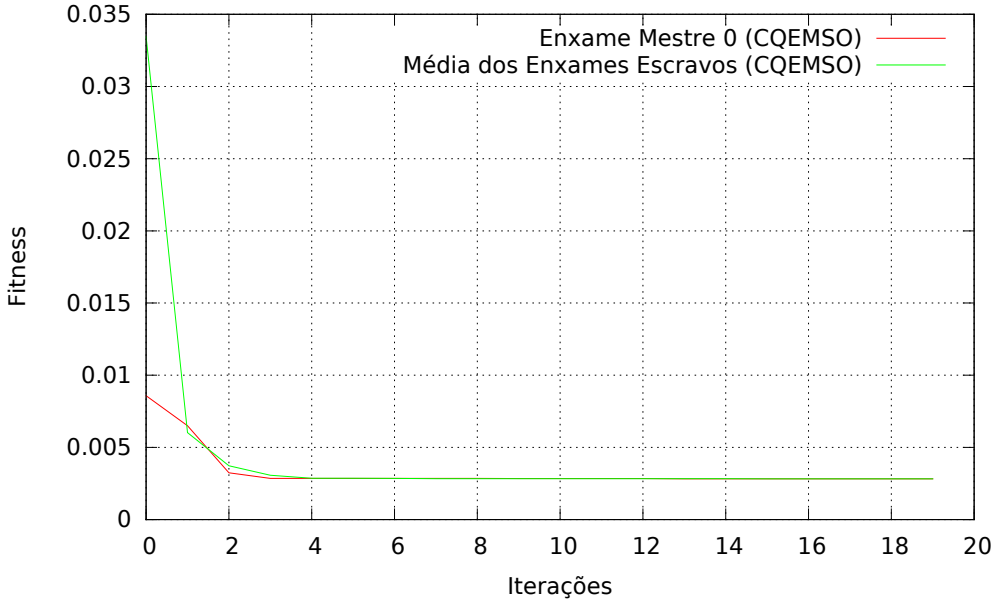


Figura 4.49: Convergência do enxame mestre do algoritmo CQEMSO para o MWTCS



As Figuras 4.50 e 4.51 mostram a média de convergência dos enxames escravos dos algoritmos COMSO, COEMSO, CEMSO, COQMSO e CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 20 iterações.

Figura 4.50: Média de convergência dos enxames escravos para os algoritmo COMSO, CEMSO e COEMSO no problema MWTCS

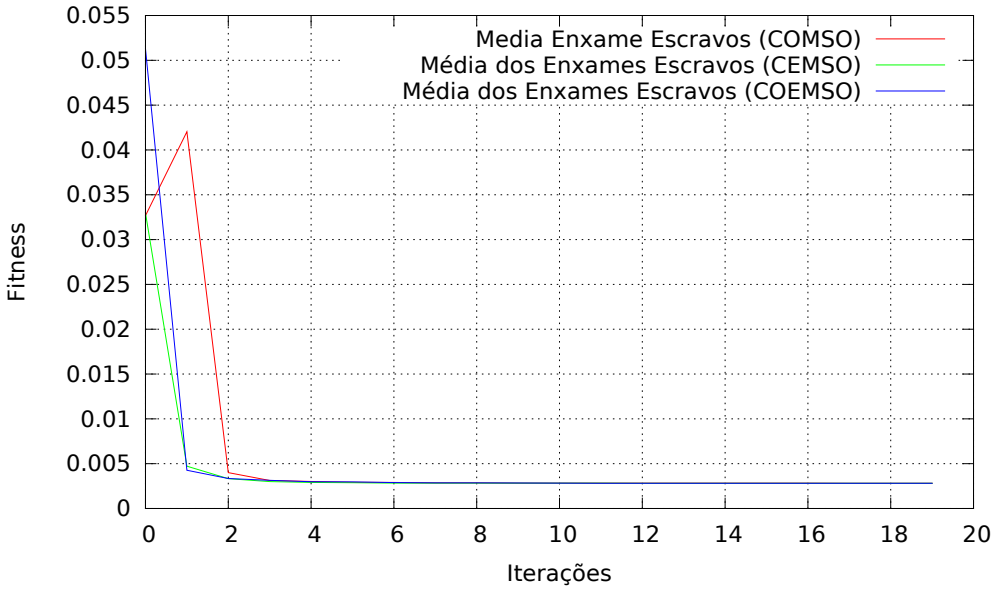
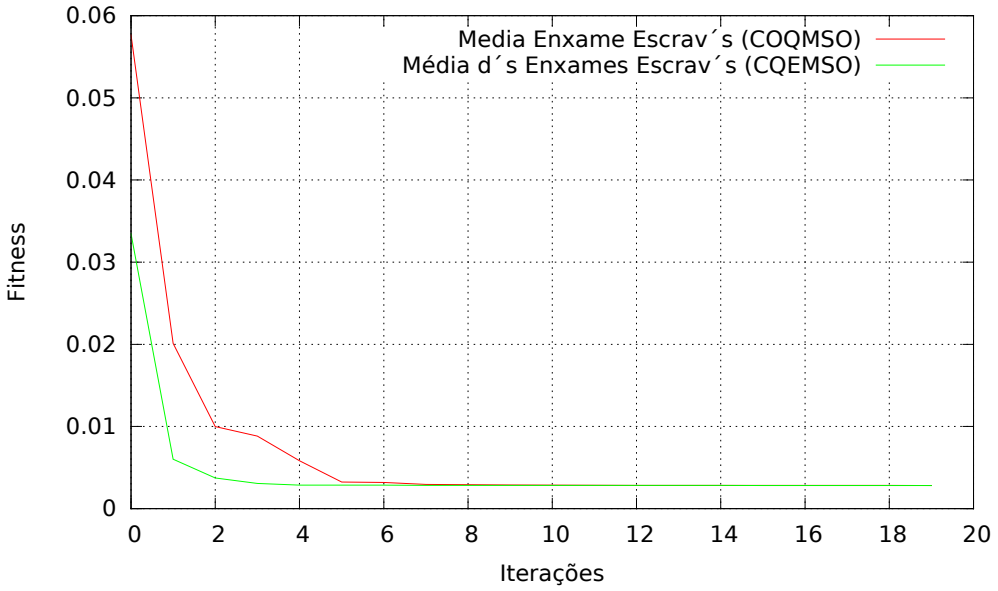


Figura 4.51: Média de convergência dos enxames escravos para os algoritmo COQMSO e CQEMSO no problema MWTCS



Análise dos Resultados para o MWTCS

Com os resultados obtidos pelos algoritmos CEMSO e CQEMSO, é possível estabelecer uma comparação dos resultados obtidos nessas simulações com resultados presentes na literatura. Foram utilizados os melhores resultados do experimento de 80 iterações.

Segue-se os algoritmos usados para comparação:

- ***Alg. 1 [Teixeira 2012]***: Algoritmo genético com interação social baseado na teoria dos jogos (SIGA);
- ***Alg. 2 [He & Wang 2007]***: Otimização por enxame de partículas com mecanismos co-evolutivos aplicados em um fator de penalidade para otimização restritiva;
- ***Alg. 3 [Coello & Montes 2002]***: Algoritmo genético com mecanismo de manipulação de restrição.

Tabela 4.21: Comparação dos resultados para o problema MWTCS

Variaveis	CEMSO	CQEMSO	Alg. 1	Alg. 2	Alg. 3
d	0.050000	0.050000	0.500250	0.051728	0.050000
D	0.282023	0.282023	0.280748	0.357644	0.315900
P	2.000000	2.000000	2.036163	11.244543	14.250000
G_1	-0.000000	-0.000000	-0.002840	-0.000020	-0.000014
G_2	-0.235327	-0.235327	-0.249450	-0.035891	-0.003782
G_3	-43.146145	-43.146145	-42.176000	-27.886075	-3.938302
G_4	-0.778651	-0.778651	-0.780140	-63.345953	-0.756067
Violações	0	0	0	0	0
Tempo	0.056106	0.065769	N/A	N/A	N/A
Fitness	0.002820	0.002820	0.002836	0.0126747	0.0128334

Através dos resultados obridos pelos experimentos, pode-se constatar que:

- O melhor resultado encontrado para o problema MWTCS para os algoritmos com base no PSO para as partículas originais foi 0.002820 e foram obtidos pelos seguintes algoritmos:
  1. CEMSO: Obteve o melhor resultado em todos os experimentos;
  2. COMSO: Obteve o melhor resultado em todos os experimentos;
  3. PSO+EE: Obteve o melhor resultado em todos os experimentos;
  4. PSO: Obteve o melhor resultado a partir do experimento de 8 iterações.

- O melhor resultado encontrado para o problema MWTCS para os algoritmos com base no EPSO para as partículas originais foi 0.002820 e foram obtidos pelos seguintes algoritmos:

    1. COEMSO: Obteve o melhor resultado em todos os experimentos;
    2. EPSO: Obteve o melhor resultado a partir do experimento de 8 iterações.

- O melhor resultado encontrado para o problema MWTCS para os algoritmos com base no QPSO para as partículas originais foi 0.002820 e foram obtidos pelos seguintes algoritmos:

    1. CQEMSO: Obteve o melhor resultado em todos os experimentos;
    2. COQMSO: Obteve o melhor resultado a partir do experimento de 8 iterações;
    3. QPSO+EE: Obteve o melhor resultado em todos os experimentos;
    4. QPSO: Obteve o melhor resultado a partir do experimento de 10 iterações.

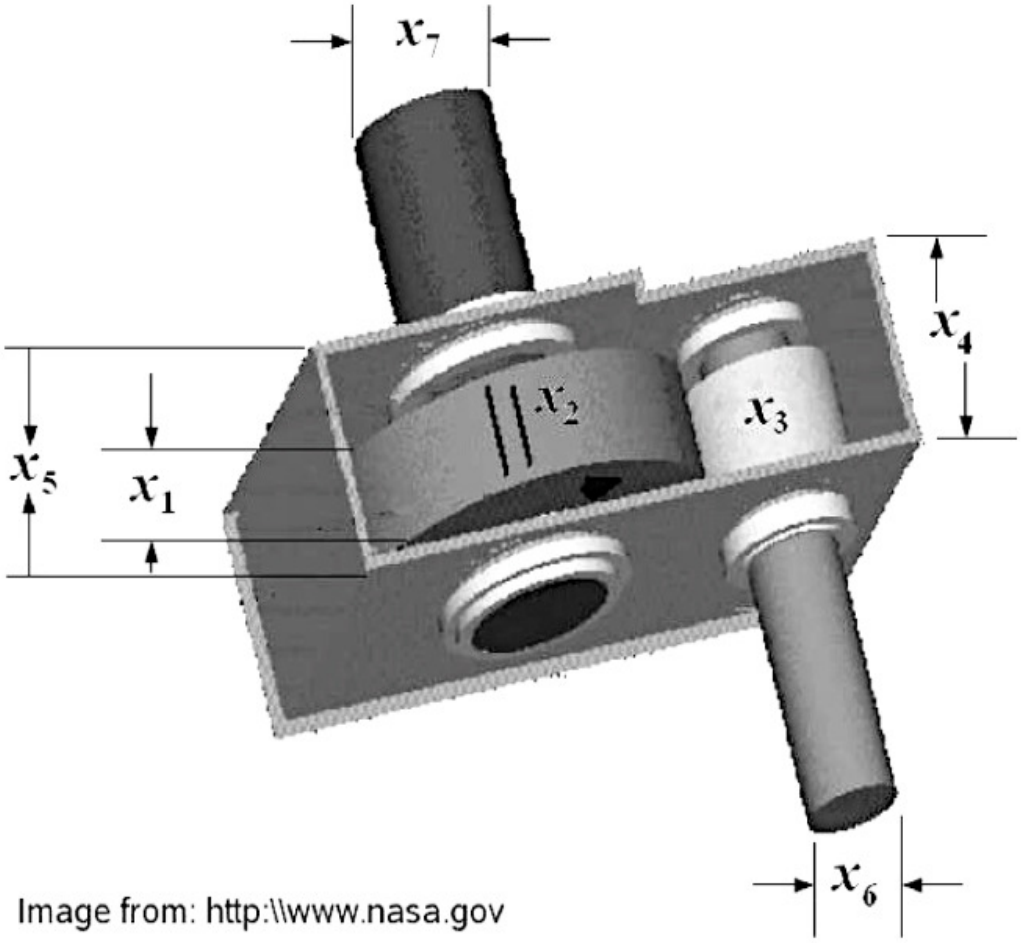
- Os algoritmos que utilizam a abordagem multi-enxame de topologia mestre-escravos para o problema MWTCS obtiveram os melhores valores de média e variância em todos os experimentos realizados, quando comparados com as suas respectivas versões de apenas um enxame;
- O algoritmo CQEMSO possui o melhor desempenho em comparação ao algoritmo COQMSO em todos os experimentos realizados;
- Os gráficos de convergência do algoritmo CQEMSO indicam uma velocidade de convergência elevada, encontrando o melhor resultado a partir da quarta iteração (Enxame escravo 3);
- O comportamento de convergência observado no algoritmo CQEMSO (Gráficos 4.48 e 4.49) demonstram um melhor desempenho no processo de busca entre as iterações, encontrando o melhor resultado a partir da iteração 2 nos enxames escravos e iteração 3 no enxame mestre;
- O algoritmo CEMSO apresentou média e variância do melhor resultado encontrado significativamente superior, bem como a melhor média e variância em todos os experimentos;
- Ao serem comparados com os resultados de outras soluções encontradas na literatura (expostos na Tabela 4.21), os algoritmos CEMSO e CQEMSO obtiveram os melhores resultados.

4.1.4 Projeto de Redutor de Velocidade

Projeto de Redutor de Velocidade (SRD) consiste em um sistema de redução de velocidade através da minimização do peso, obedecendo certas restrições como: Flexão de estresse dos dentes da engrenagem, tensão superficial, desvios transversais das hastes e as tensões no eixo [Teixeira et al. 2011, Souza et al. 2013, Cagnina et al. 2008]. As variáveis do problema são: Largura do rosto (X_1 ou b); módulo de dentes (X_2 ou m); número de dentes do pinhão (X_3 ou z); comprimento do primeiro eixo entre os rolamentos (X_4ou l_1); comprimento do segundo eixo entre os rolamentos (X_5 ou l_2); diâmetro do primeiro eixo (X_6 ou d_1); diâmetro do segundo eixo (X_7 ou d_2).

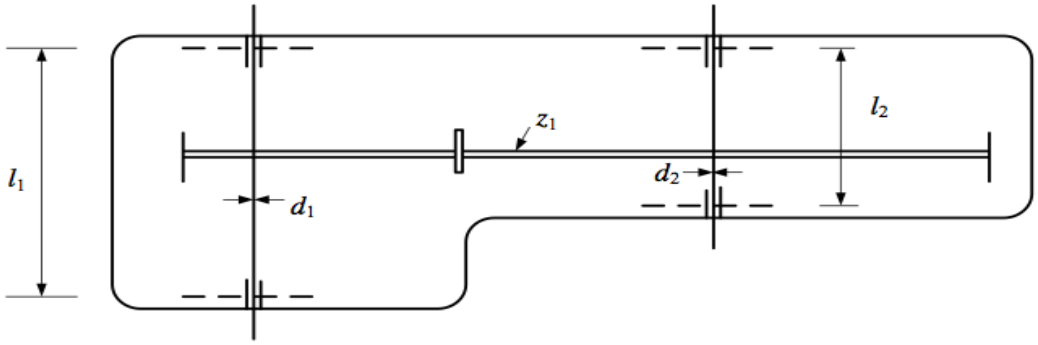
A Figura 4.52 mostra a estrutura de um redutor de velocidade.

Figura 4.52: Estrutura de um redutor de velocidade. Fonte:[Cagnina et al. 2008]



A Figura 4.53 mostra o esquema do SRD.

Figura 4.53: Esquema do *Speed Reducer Design* (SRD). Fonte:[Cagnina et al. 2008, Brajevic et al. 2010]



O SRD pode ser formalizado matematicamente através do seguinte esquema:

$$\begin{aligned}\text{Minimizar } f(X_i) = {} & 0.7854X_1X_2^2(3.3333X_3^2 + 14.933X_3 - 43.0934) \\ & - 1.508X_1(X_6^2 + X_7^2) + 7.4777(X_6^2 + X_7^2) \\ & + 0.78054(X_4X_6^2 + X_5X_7^2)\end{aligned} \tag{4.32}$$

Sujeito a:

$$g_1 = \frac{27}{X_1x_2^2X_3} - 1 \leq 0 \tag{4.33}$$

$$g_2 = \frac{397.5}{X_1X_2^2X_3^2} - 1 \leq 0 \tag{4.34}$$

$$g_3 = \frac{1.93X_4^3}{X_2X_3X_6^4} - 1 \leq 0 \tag{4.35}$$

$$g_4 = \frac{1.93X_4^3}{X_2X_3X_7^4} - 1 \leq 0 \tag{4.36}$$

$$g_5 = \frac{1.0}{110X_6^3}\sqrt{\left(\frac{750.0X_4}{X_2X_3}\right)^2 + 16.910^6} - 1 \leq 0 \tag{4.37}$$

$$g_6 = \frac{1.0}{85X_7^3}\sqrt{\left(\frac{750.0X_5}{X_2X_3}\right)^2 + 157.510^6} - 1 \leq 0 \tag{4.38}$$

$$g_7 = \frac{X_2X_3}{40} - 1 \leq 0 \tag{4.39}$$

$$g_8 = \frac{5X_2}{X_1} - 1 \leq 0 \tag{4.40}$$

$$g_9 = \frac{X_1}{12X_2} - 1 \leq 0 \tag{4.41}$$

$$g_{10} = \frac{1.5X_6 + 1.9}{X_4} - 1 \leq 0 \tag{4.42}$$

$$g_{11} = \frac{1.1X_7 + 1.9}{X_4} - 1 \leq 0 \tag{4.43}$$

Os limites de variáveis X_1 (b), X_2 (m), X_3 (z), X_4 (l_1), X_5 (l_2), X_6 (d_1), X_7 (d_2) são: $2.6 \leq X_1(d) \leq 3.6$; $0.7 \leq X_2(m) \leq 0.8$; $17 \leq X_3(z) \leq 28$; $7.3 \leq X_4(l_1) \leq 8.3$; $7.8 \leq X_5(l_2) \leq 8.3$; $2.9 \leq X_6(d_1) \leq 3.9$; $5.0 \leq X_7(d_2) \leq 5.5$.

No intuito de avaliar os algoritmos CEMSO e CQEMSO, foram utilizadas as seguintes configurações: Iterações = 20, 40, 80, 200, 400 e 1000; quantidade de execuções = 500; quantidade de partículas para cada enxame escravo = 80; quantidade de réplicas por partícula = 4; quantidade de enxames escravos = 4; constante de pertubação para o melhor global (σ_g) = 0.005; parâmetro de estratégias para a mutação dos fatores de inércia e de aceleração (σ) = 0.22; fatores C_1 e C_2 (COEMSO e CEMSO) = 2.05; fator de aceleração para o enxame mestre (C_3) = 2.02; quantidade de testes executados para cada algoritmo = 500; Fator de diminuição de velocidade (δ) = diminuir a velocidade a 40% (0.4); Fator de penalidade para resultados com violações = 200; Constante de probabilidade (θ) = 0.5; Velocidade mínima (V_{MIN}) = metade do valor de limite máximo de posição com sinal ou direção invertida ($-(\frac{X_{MAX}}{2})$). Os valores de fator de inércia (w) (CEMSO), bem como fatores α e β (CQEMSO e COQMSO) são gerados a cada iteração pelas Equações 2.2, 2.30 e 2.32 (decrescimento randômico).

Além dos algoritmos CEMSO e CQEMSO, foram utilizados para efeito de comparação os algoritmos PSO, EPSO, QPSO, PSO+EE e QPSO+EE baseados na implementação exposta nos capítulos 3 e 4, bem como suas versões multi-enxame (COMSO, COEMSO e COQMSO). As Tabelas 4.22, 4.23, 4.24, 4.25, 4.26 e 4.27 mostram os resultados obtidos nos experimentos realizados para o problema SRD.

Tabela 4.22: Comparação dos resultados para 20 iterações para o SRD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	3281.0273037	2895.371338	66	$1.441843e+05$	0.005525 s
EPSO	2959.5638584	2897.608643	5	$1.878563e+03$	0.018772 s
QPSO	3035.3324590	2907.956543	0	$2.825707e+03$	0.004012 s
PSO+EE	2927.2141841	2894.930176	5	$1.609805e+03$	0.015828 s
QPSO+EE	2978.7923589	2900.748779	0	$2.366843e+03$	0.020574 s
COMSO	2932.0128096	2895.056641	6	$1.283687e+03$	0.007525 s
COEMSO	2906.0073965	2895.439209	0	$3.593382e+01$	0.030661 s
COQMSO	2980.7902935	2907.956543	0	$1.897609e+03$	0.008247 s
CEMSO	2895.3799849	2894.921143	0	$2.069207e-01$	0.030733 s
CQEMSO	2939.3739272	2900.256836	0	$3.850386e+02$	0.034645 s

Tabela 4.23: Comparação dos resultados para 40 iterações para o SRD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	3140.3694546	2894.904541	43	$6.106735e+04$	0.007671 s
EPSO	2927.1337603	2895.803223	3	$1.002012e+03$	0.028131 s
QPSO	2985.1412271	2904.233887	0	$2.462765e+03$	0.009431 s
PSO+EE	2917.0838364	2894.901855	2	$1.515971e+04$	0.027463 s
QPSO+EE	2936.7109614	2897.001709	0	$4.725249e+02$	0.029359 s
COMSO	2917.3586680	2894.903809	4	$5.857949e+02$	0.017489 s
COEMSO	2898.5336270	2895.053467	0	$5.225132e+00$	0.063476 s
COQMSO	2945.6316465	2898.089844	0	$5.841100e+02$	0.015654 s
CEMSO	2894.9374985	2894.901611	0	$5.428871e-03$	0.062229 s
CQEMSO	2917.6278750	2896.047363	0	$1.371094e+02$	0.064520 s

Tabela 4.24: Comparação dos resultados para 80 iterações para o SRD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	3156.5580327	2894.901611	46	$6.997685e+04$	0.013998 s
EPSO	2906.6872896	2895.006836	1	$2.014692e+02$	0.052833 s
QPSO	2958.0506772	2901.244141	0	$1.269971e+03$	0.019204 s
PSO+EE	2900.8456978	2894.901611	1	$5.080360e+01$	0.054280 s
QPSO+EE	2909.6530107	2895.275635	0	$1.194643e+02$	0.082140 s
COMSO	2927.5168262	2894.901611	6	$1.268792e+03$	0.033793 s
COEMSO	2896.0558306	2894.968750	0	$1.375134e+00$	0.124154 s
COQMSO	2931.5126328	2895.840332	0	$3.119054e+02$	0.031609 s
CEMSO	2894.9041187	2894.901611	0	$2.034531e-04$	0.122744 s
CQEMSO	2901.5355493	2894.945801	0	$2.785339e+01$	0.140049 s

Tabela 4.25: Comparação dos resultados para 200 iterações para o SRD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	3132.8222197	2894.901611	42	$5.812468e+04$	0.056037 s
EPSO	2896.9522563	2894.901855	0	$2.969861e+01$	0.177428 s
QPSO	2927.7451367	2896.704346	0	$4.714859e+02$	0.040539 s
PSO+EE	2894.9081133	2894.901611	0	$1.285163e-03$	0.176115 s
QPSO+EE	2895.5255469	2894.901611	0	$1.209680e+00$	0.004855 s
COMSO	2924.4426670	2894.901611	5	$2.842573e+03$	0.078592 s
COEMSO	2895.1229355	2894.901611	0	$1.133051e-01$	0.299733 s
COQMSO	2913.9304448	2895.415771	0	$1.335161e+02$	0.082136 s
CEMSO	2894.9016445	2894.901611	0	$5.121365e-08$	0.304501 s
CQEMSO	2895.0724375	2894.901611	0	$7.951790e-02$	0.344892 s

Tabela 4.26: Comparação dos resultados para 400 iterações para o SRD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	3095.5353398	2894.901611	35	$4.088480e+04$	0.063694 s
EPSO	2896.3858057	2894.901611	0	$2.279394e+01$	0.336074 s
QPSO	2913.8420962	2895.558838	0	$1.768186e+02$	0.089659 s
PSO+EE	2894.9018291	2894.901611	0	$7.245928e-07$	0.388315 s
QPSO+EE	2894.9180020	2894.901611	0	$1.737085e-03$	0.349189 s
COMSO	2927.6522441	2894.901611	6	$1.248648e+03$	0.150385 s
COEMSO	2895.0374067	2894.901611	0	$8.419339e-02$	0.578920 s
COQMSO	2905.9240898	2895.079102	0	$6.288676e+01$	0.165805 s
CEMSO	2894.9016221	2894.901611	0	$5.140094e-09$	0.594382 s
CQEMSO	2894.9041606	2894.901611	0	$3.806591e-05$	0.701476 s

Tabela 4.27: Comparação dos resultados para 1000 iterações para o SRD

Algoritmo	Média	Melhor *Fitness*	Violações	Variância	Tempo (Média)
PSO	3081.7047524	2894.901611	33	$3.601403e+04$	0.169510 s
EPSO	2896.5815850	2894.901611	0	$3.611550e+01$	0.653707 s
QPSO	2903.5802314	2895.331787	0	$5.257331e+01$	0.196143 s
PSO+EE	2894.9016113	2894.901611	0	$0.000000e+00$	0.792240 s
QPSO+EE	2894.9016162	2894.901611	0	$6.426312e-09$	1.023685 s
COMSO	2910.6380586	2894.901611	3	$4.089929e+02$	0.359374 s
COEMSO	2894.9946182	2894.902100	0	$5.651377e-02$	1.290117 s
COQMSO	2899.5233306	2895.239746	0	$1.848147e+01$	0.390909 s
CEMSO	2894.9016113	2894.901611	0	$0.000000e+00$	1.501839 s
CQEMSO	2894.9016113	2894.901611	0	$0.000000e+00$	1.744367 s

As Figuras 4.54 e 4.55 mostram, respectivamente, o comparativo entre os melhores valores obtidos e a média dos resultados encontrados de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.54: Gráfico comparativo dos melhores resutados obtidos para o SRD

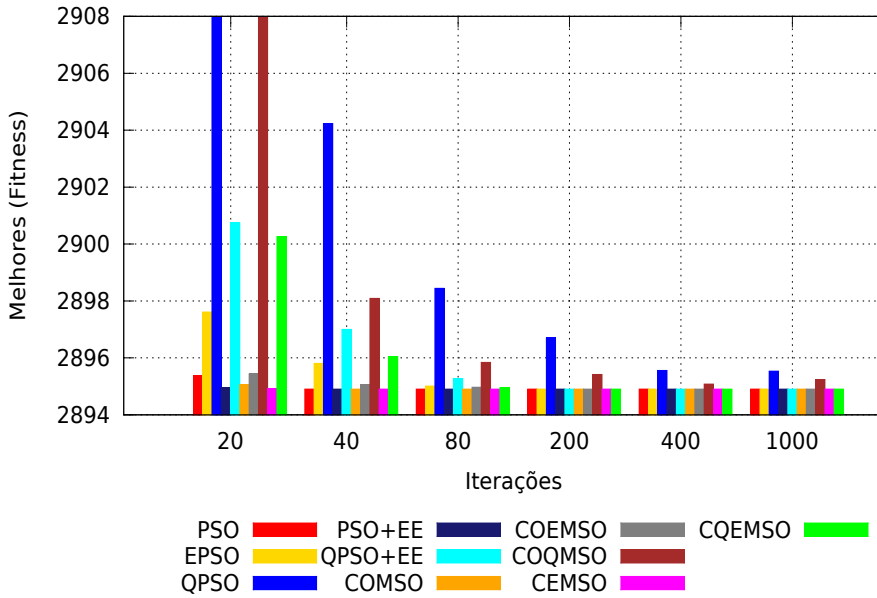
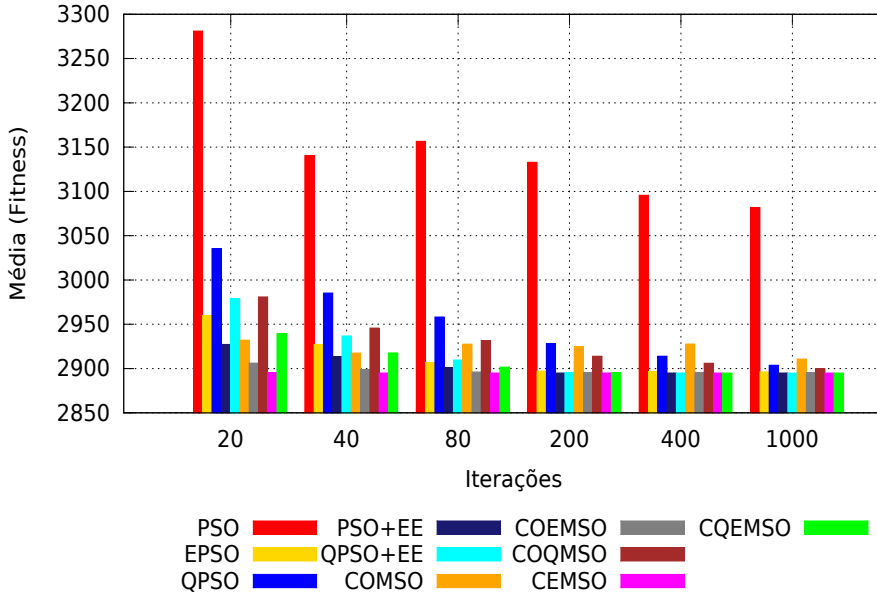


Figura 4.55: Gráfico comparativo das médias para o SRD



As Figuras 4.56 e 4.57 mostram, respectivamente, a varância dos resultados encontrados e o tempo de execução de cada algoritmo para todas as quantidades de iterações utilizadas nos experimentos.

Figura 4.56: Gráfico comparativo dos valores de variância para o SRD

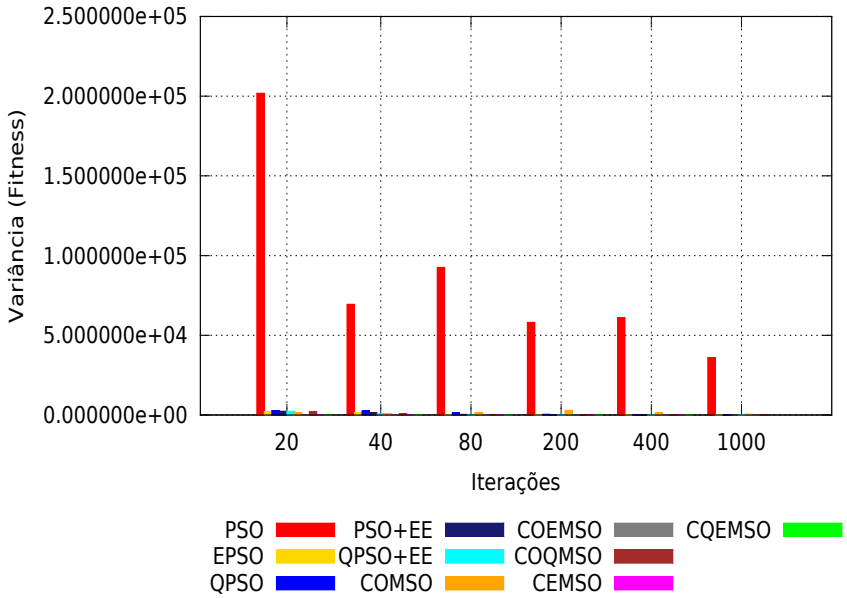
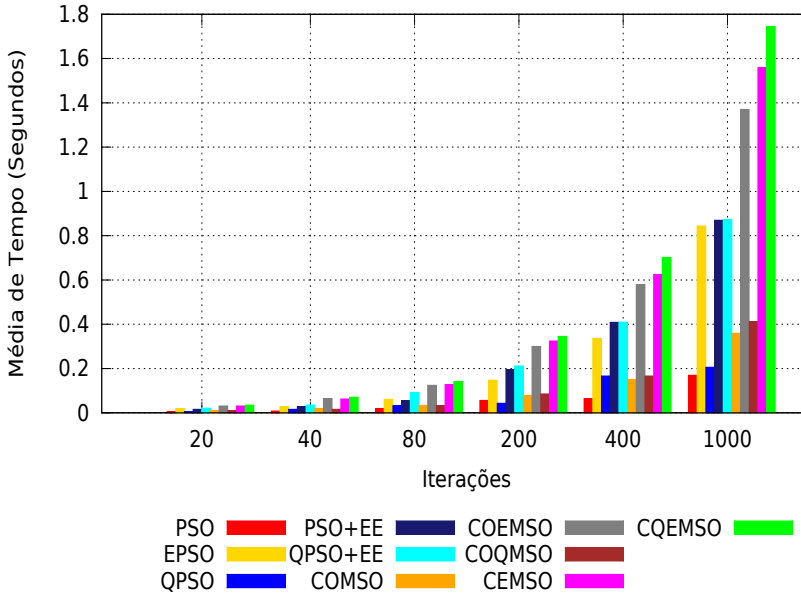


Figura 4.57: Gráfico comparativo da média do tempo de execução dos algoritmos testados para o SRD



As Figuras 4.58 e 4.59 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.58: Convergência dos enxames escravos do algoritmo COMSO para o SRD

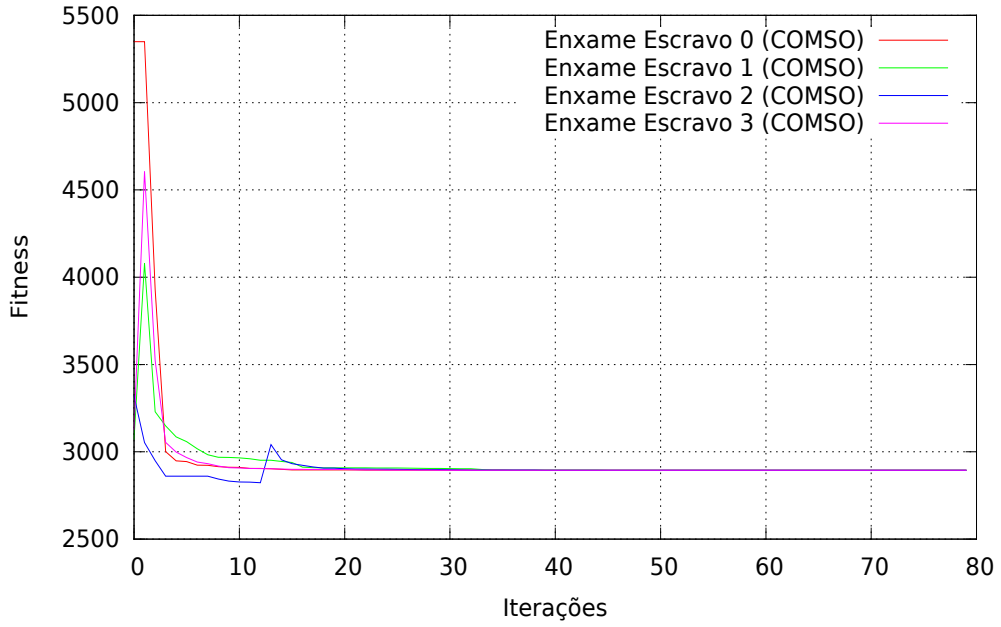
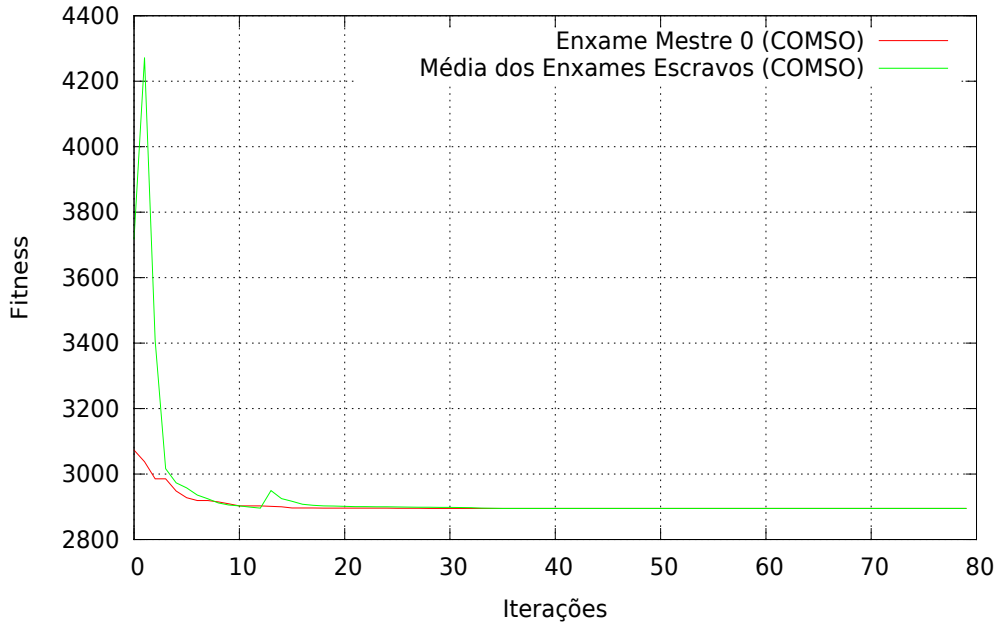


Figura 4.59: Convergência do enxame mestre do algoritmo COMSO para o SRD



As Figuras 4.60 e 4.61 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.60: Convergência dos enxames escravos do algoritmo COEMSO para o SRD

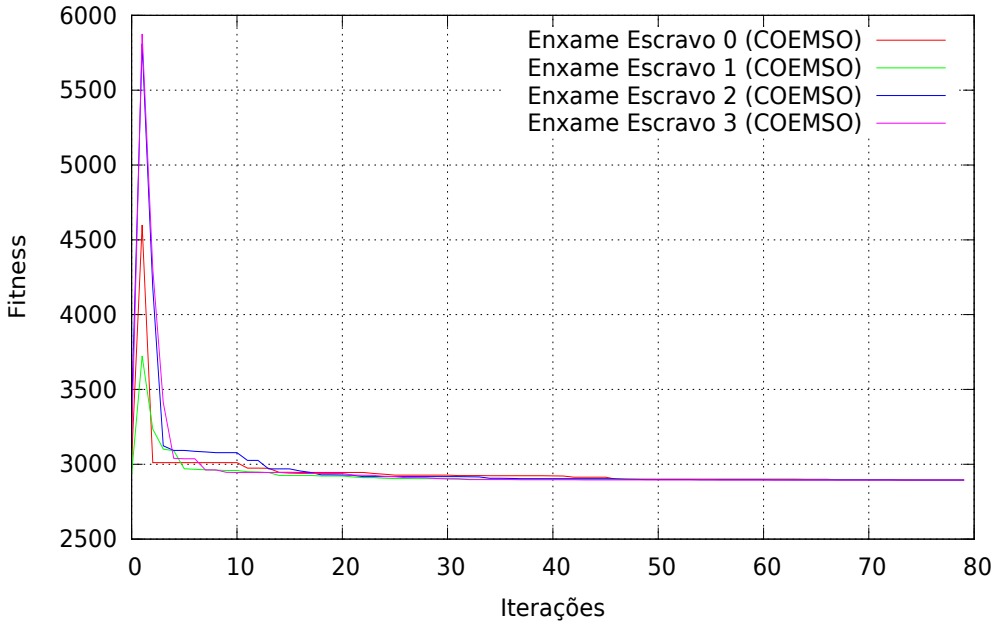
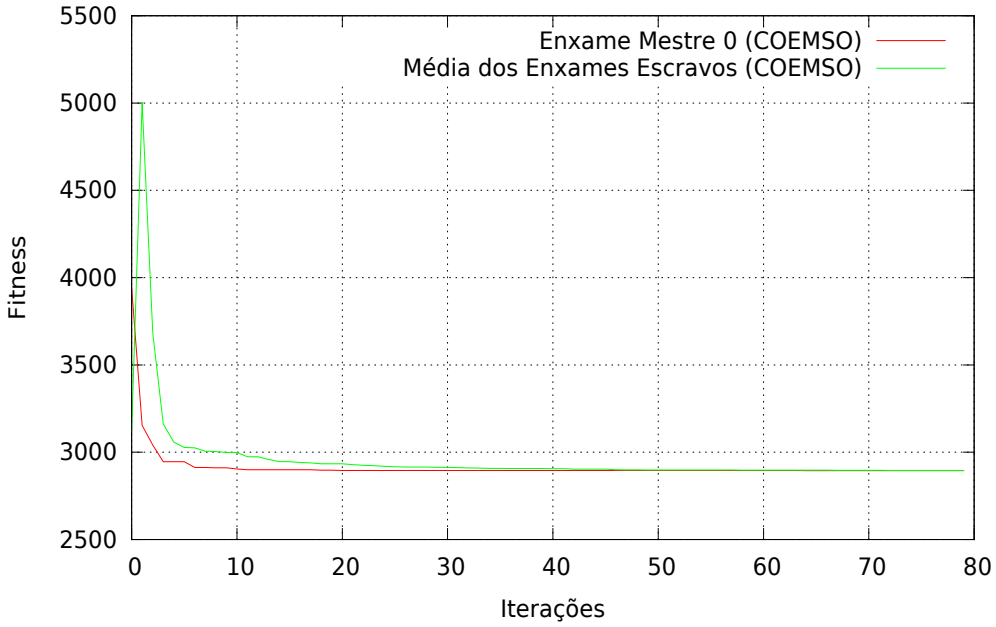


Figura 4.61: Convergência do enxame mestre do algoritmo COEMSO para o SRD



As Figuras 4.62 e 4.63 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo COQMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.62: Convergência dos enxames escravos do algoritmo COQMSO para o SRD

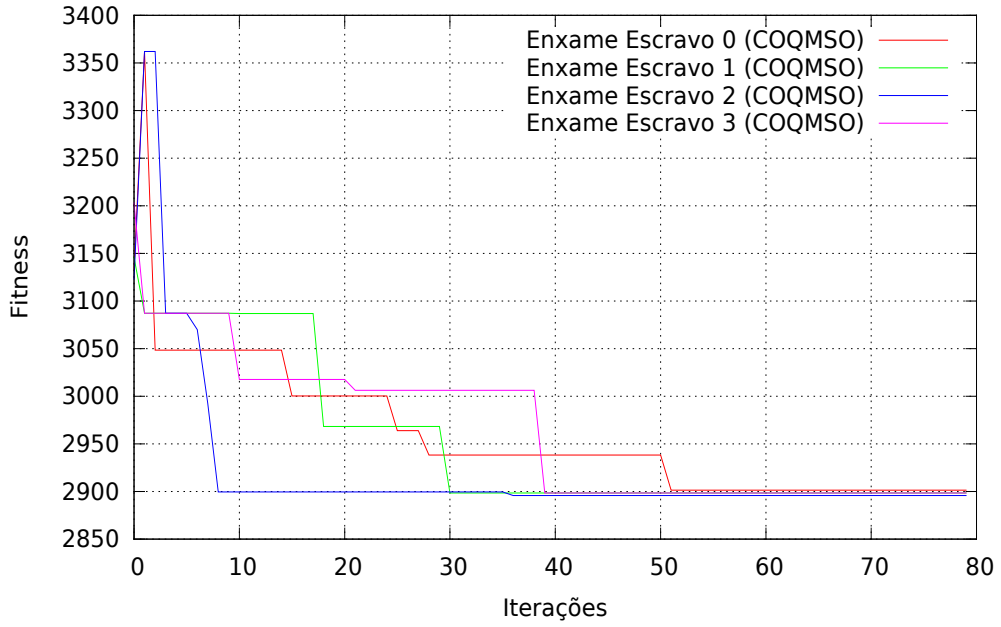
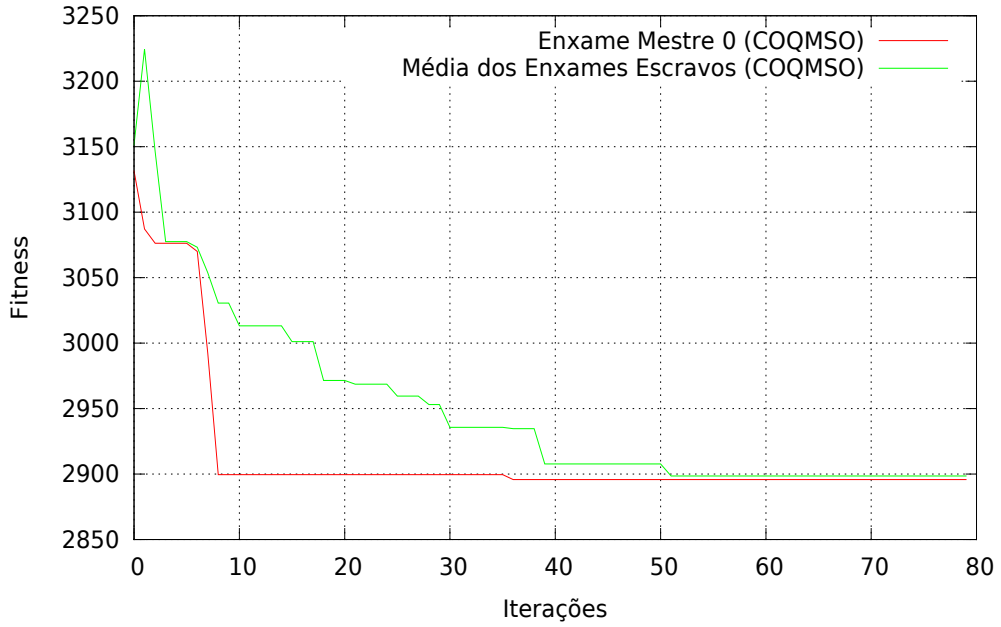


Figura 4.63: Convergência do enxame mestre do algoritmo COQMSO para o SRD



As Figuras 4.64 e 4.65 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.64: Convergência dos enxames escravos do algoritmo CEMSO para o SRD

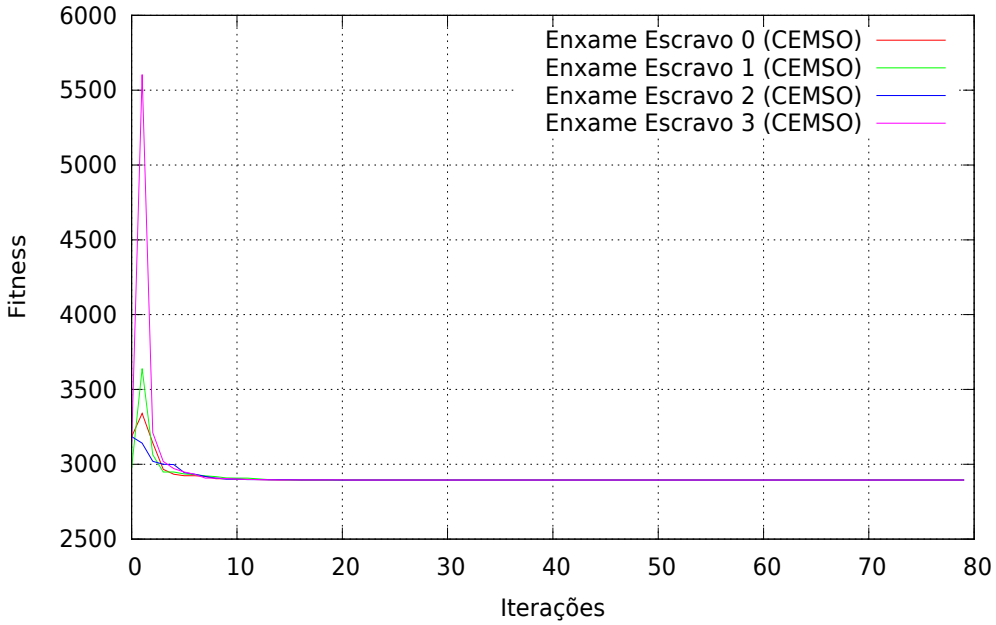
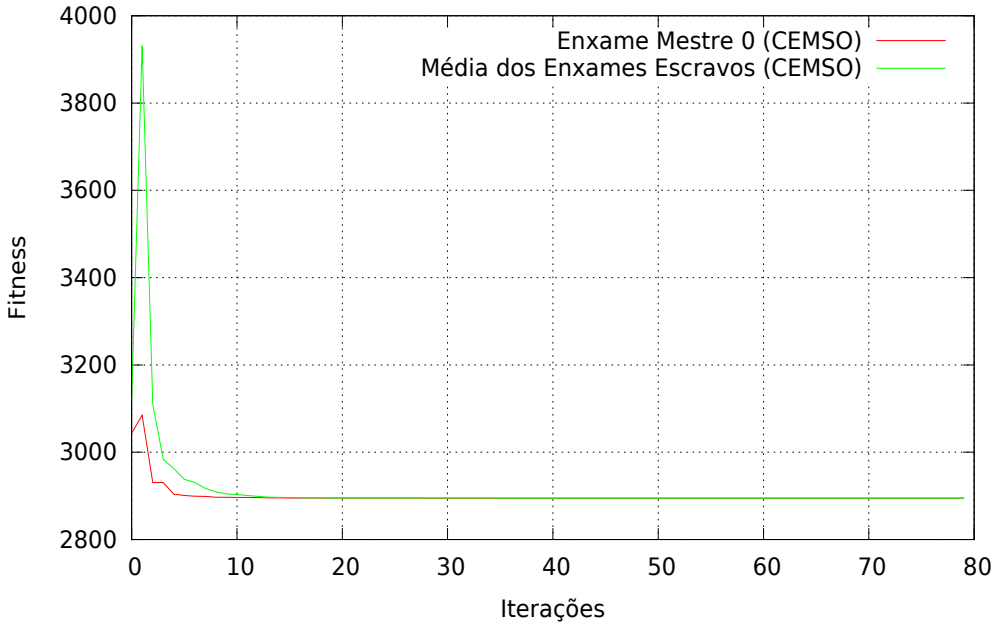


Figura 4.65: Convergência do enxame mestre do algoritmo CEMSO para o SRD



As Figuras 4.66 e 4.67 mostram o comportamento de convergência dos enxames escravos e mestre do algoritmo CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.66: Convergência dos enxames escravos do algoritmo CQEMSO para o SRD

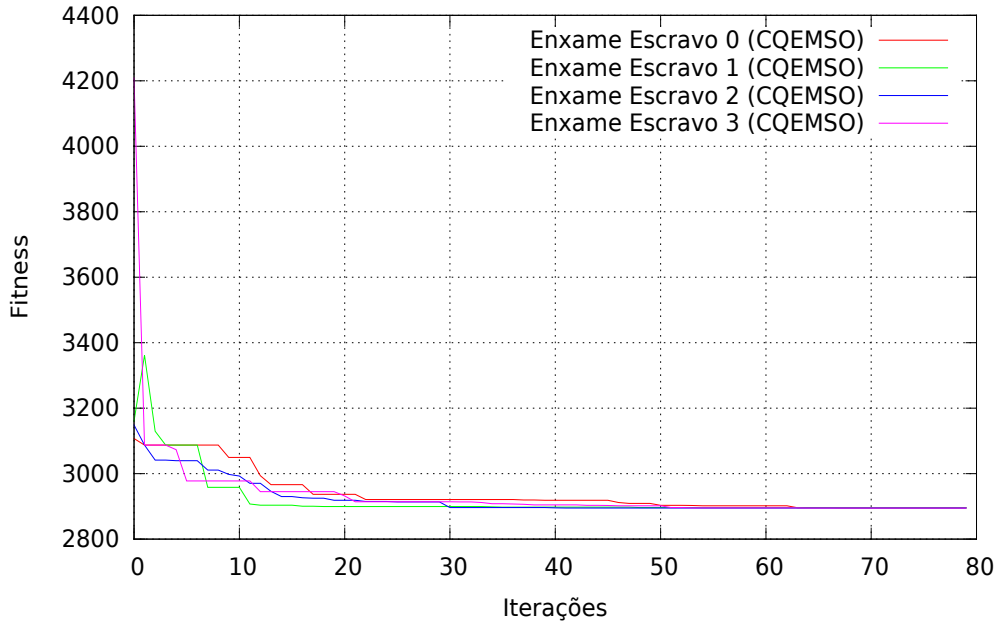
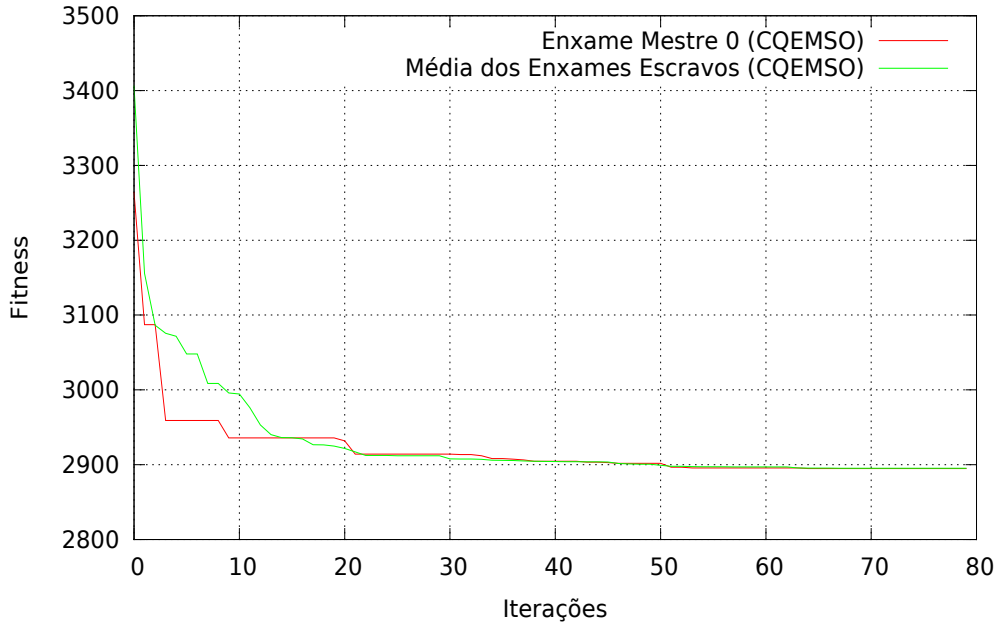


Figura 4.67: Convergência do enxame mestre do algoritmo CQEMSO para o SRD



As Figuras 4.68 e 4.69 mostram a média de convergência dos enxames escravos dos algoritmos COMSO, COEMSO, CEMSO, COQMSO e CQEMSO, obtidos nos melhores resultados encontrados para o experimento de 80 iterações.

Figura 4.68: Média de convergência dos enxames escravos para os algoritmo COMSO, CEMSO e COEMSO para o SRD

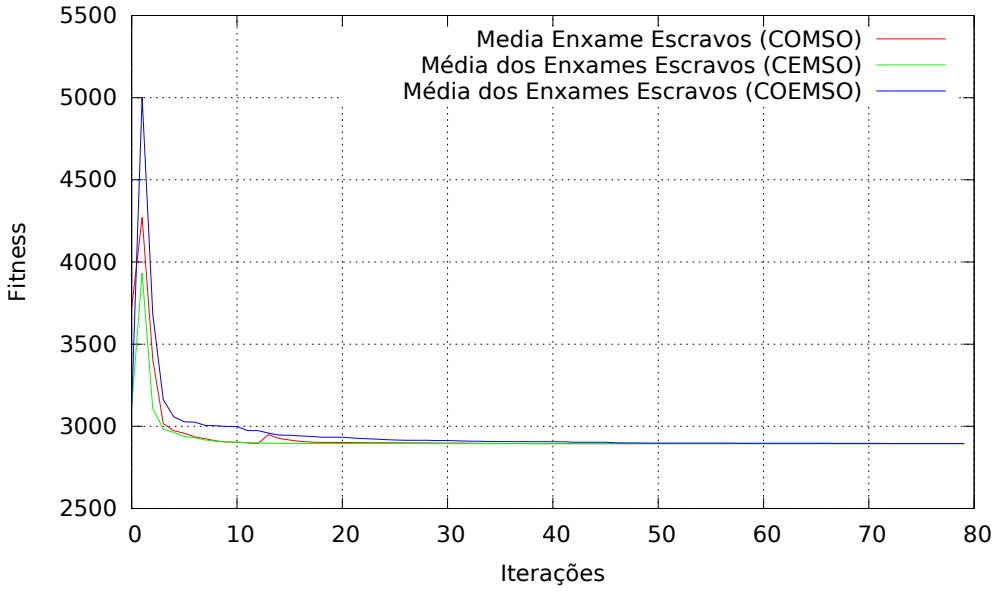
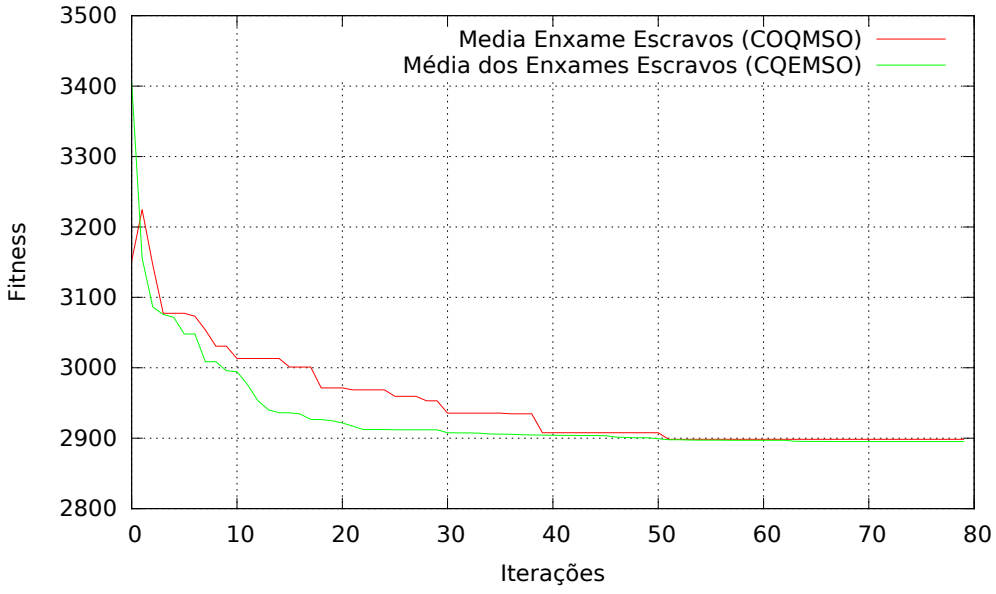


Figura 4.69: Média de convergência dos enxames escravos para os algoritmo COQMSO e CQEMSO para o SRD



Análise dos Resultados para o SRD

Com os resultados obtidos pelos algoritmos CEMSO e CQEMSO, é possível estabelecer uma comparação dos resultados obtidos nessas simulações com resultados presentes na literatura. Foram utilizados os melhores resultados do experimento de 1000 iterações.

Segue-se os algoritmos usados para comparação:

- ***Alg. 1 [Teixeira et al. 2011]***: Algoritmo genético com interação social baseado em teoria dos jogos (SIGA);
- ***Alg. 2 [Brajevic et al. 2010]***: Otimização por colônia de abelhas modificado para otimização restritiva;
- ***Alg. 3 [Cagnina et al. 2008]***: Otimização por enxame de partículas adapatado para problemas de engenharia com equação de velocidade modificada.

Tabela 4.28: Comparação dos resultados para o problema SRD

Variaveis	CEMSO	CQEMSO	Alg. 1	Alg. 2	Alg. 3
X_1 (b)	3.500000	3.500204	3.500459	3.500000	3.500000
X_2 (m)	0.700000	0.700000	0.700020	0.700000	0.700000
X_3 (z)	17.000000	17.000000	17.005030	17.000000	17.000000
X_4 (l_1)	7.300000	7.300000	7.300251	7.300000	7.300000
X_5 (l_2)	7.800000	7.800000	7.800195	7.800000	7.800000
X_6 (d_1)	2.900000	2.900000	2.900041	3.350215	3.350214
X_7 (d_2)	5.286684	5.286830	5.286863	5.286683	5.286683
G_1	-0.073915	-0.073969	-0.074364	-0.073915	-0.073915
G_2	-0.197999	-0.198045	-0.198624	-0.197996	-0.197998
G_3	-0.107955	-0.107955	-0.108202	-0.499172	-0.499172
G_4	-0.901472	-0.901483	-0.901443	-0.901471	-0.901471
G_5	-1.000000	-1.000000	-1.000000	-2.220e-16	-0.000000
G_6	-0.000000	-0.000084	-0.000102	-3.331e-16	-5.000e-16
G_7	-0.702500	-0.702500	-0.702403	-0.702500	-0.702500
G_8	-0.000000	-0.000058	-0.000103	-0.000000	-1.000e-16
G_9	-0.795833	-0.795821	-0.795801	-0.583333	-0.583333
G_{10}	-0.143836	-0.143836	-0.143857	-0.0513265	-0.051325
G_{11}	-0.010852	-0.010832	-0.011074	-0.010852	-0.010852
Violações	0	0	0	0	0
Tempo (Média)	1.501839	1.744367	N/A	N/A	N/A
Fitness	2894.901611	2894.901611	2897.531422	2996.348165	2996.348165

Através dos resultados obridos pelos experimentos, pode-se constatar que:

- O melhor resultado encontrado pelos algoritmos com base no PSO para as partículas originais foi 2894.901611 e foram obtidos pelos seguintes algoritmos:
  1. CEMSO: Obteve o melhor resultado a partir de 40 iterações;
  2. COMSO: Obteve o melhor resultado a partir de 80 iterações;
  3. PSO: Obteve o melhor resultado a partir de 80 iterações;
  4. PSO+EE: Obteve o melhor resultado a partir de 80 iterações.
- O melhor resultado encontrado pelos algoritmos com base no EPSO para as partículas originais foi 2894.901611 e foram obtidos pelos seguintes algoritmos:
  1. COEMSO: Obteve o melhor resultado a partir de 200 iterações;
  2. EPSO: Obteve o melhor resultado a partir de 400 iterações.
- O melhor resultado encontrado pelos algoritmos com base no QPSO para as partículas originais foi 2894.901611 e foram obtidos pelos seguintes algoritmos:
  1. CQEMSO: Obteve o melhor resultado a partir de 200 iterações;
  2. COQMSO: Melhor resultado obtido for 2895.079102 (400 iteração);
  3. QPSO: Melhor resultado obtido for 2895.331787 (1000 iteração);
  4. QPSO+EE: Obteve o melhor resultado a partir de 200 iterações.
- Os algoritmos que utilizam a abordagem multi-enxame de topologia mestre-escravos para o problema SRD obtiveram os melhores resultados em todos os experimentos realizados, quando comparados com as suas respectivas versões de apenas um enxame;
- O algoritmo CQEMSO obteve os melhores resultados, incluindo média e variância em comparação ao algoritmo COQMSO;
- O Figura 4.69 demonstra uma convergência mais rápida dos enxames escravos do algortimo CQEMSO em relação ao algoritmo COQMSO;
- O algoritmo CEMSO apresentou os melhores resultados em termos de média, variância e melhores resultados em todos os experimentos realizados;
- Ao serem comparados com os resultados de outras soluções encontradas na literatura (expostos na Tabela 4.28), os algoritmos CEMSO e CQEMSO obtiveram os melhores resultados.

4.2 Comentários Gerais

Esta seção aborda os assuntos relacionados a pontos fundamentais encontrados nos experimentos, bem como limitações e métodos de medição utilizados.

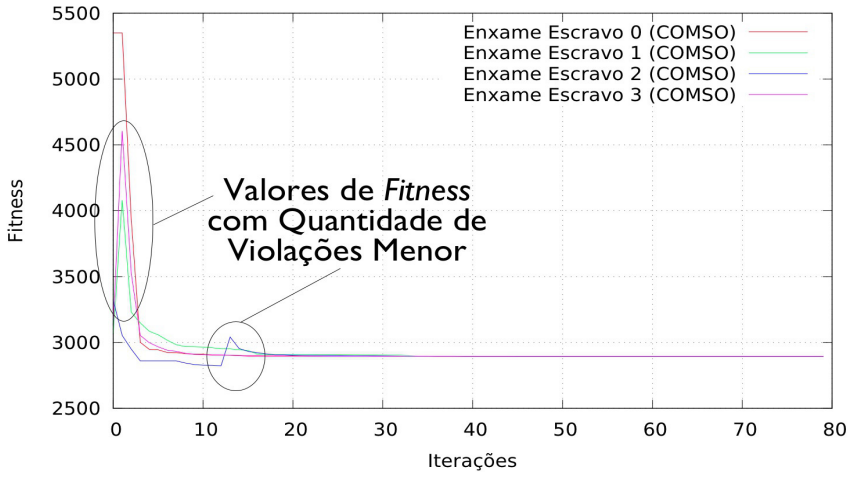
4.2.1 Seleção dos Dados de Convergência

Os experimentos documentados nesta dissertação são relacionados aos melhores resultados obtidos em 500 execuções. Tal abordagem foi selecionada devido a limitações no sistema de plotagem *Gnuplot*, o que impossibilitou a finalização do *script* que calculasse a média dos valores de convergência de todos os experimentos.

4.2.2 Picos de Máximo Global em Gráficos de Convergência

Alguns gráficos de convergência obtidos durante os experimentos apresentaram em determinadas iterações, picos de máximo global, as quais representam soluções melhores quando comparadas com as obtidas em interações anteriores. Isso ocorre devido a obtenção de um solução global onde a quantidade de violações é menor e o valor de *fitness* é maior, em relação ao resultado anterior. De maneira geral, os gráficos de convergência consideram apenas os valores de *fitness*, sendo que a obtenção de uma solução com uma quantidade de violações menor é automáticamente inserida no gráfico, independente do seu valor de *fitness* ser maior ou menor do que a solução anterior. A Figura 4.70 exemplifica o pico de máximo global relacionado a uma solução com uma quantidade menor de violação em um gráfico de convergência.

Figura 4.70: Exemplo de picos de máximo global nos gráficos de convergência



Capítulo 5

Considerações Finais

5.1 Sumarização e Conclusão

Este trabalho foi proposto com o objetivo de desenvolver um conjunto de novas metaheurísticas híbridas de busca e otimização, pertencentes a categoria de computação bioinspirada e aplicadas sob o contexto multi-enxame com topologia mestre-escravos. Essas técnicas foram denominadas de ***Competitive Evolutionary Multi Swarm Optimization*** e ***Competitive Quantum-Behaviour Evolutionary Multi Swarm Optimization*** ou, simplesmente, algoritmos CEMSO e CQEMSO, respectivamente.

No intuito de cumprir esse objetivo, inicialmente fora realizada uma contextualização e um paralelo entre a abordagem clássica dos enxames de partículas (PSO) de [Kennedy & Eberhart 1995], o enxame de partículas com inspiração quântica (QPSO) de [Sun et al. 2004*b*], o enxame de partículas evolucionário (EPSO) de [Miranda & Fonseca 2002], aspectos relativos às estratégias evolutivas utilizadas no algoritmo EPSO. Questões inerentes à abordagens multi-enxame com ênfase em soluções baseadas em topologia mestre-escravos [Niu et al. 2005, Niu et al. 2007] foram também analisadas.

A partir dessa contextualização, surgiu a motivação para a proposição e desenvolvimento dos algoritmos CEMSO e CQEMSO, ambos baseados em duas metaheurísticas desenvolvidas também nesta dissertação: PSO+EE e QPSO+EE. Além disso, as extensões multi-enxame de topologia mestre-escravos dos algoritmos PSO (COMSO, baseado na implementação de [Niu et al. 2007]), EPSO (COEMSO, desenvolvido nesta dissertação) e QPSO (COQMSO, proposto neste trabalho) tem as suas motivações observadas nesse mesmo processo de contextualização.

É válido ressaltar que a utilização do algoritmo PSO sob a arquitetura CUDA e os métodos de organização dos dados na GPU, que serviram como referência inicial para a implementação dos algoritmos COMSO, COEMSO, COQMSO, CEMSO e CQEMSO, foram objeto de estudo do Trabalho de Conclusão de Curso do autor [Souza 2010], tendo gerado posteriormente uma publicação [Souza et al. 2011].

Durante o processo de levantamento bibliográfico e análise das abordagens competitivas e cooperativas para algoritmos de otimização por enxame de partículas (com destaque para os modelos descritos em [den Bergh & Engelbrecht 2004, El-Abd & Kamel 2008]), os trabalhos de [Niu et al. 2005, Niu et al. 2007] foram estudados e, a partir deles, foi desenvolvido em [Souza et al. 2013] uma versão do algoritmo EPSO utilizando uma abordagem multi-enxame de topologia mestre-escravos. Este trabalho seria a base fundamental para o desenvolvimento dos algoritmos PSO+EE e QPSO+EE, onde seriam utilizados posteriormente como motor principal dos algoritmos CEMSO e CQEMSO, respectivamente.

Posteriormente a publicação de [Souza et al. 2013], alguns experimentos foram realizados no intuito de aplicar a abordagem competitiva da solução proposta por [Niu et al. 2005, Niu et al. 2007] ao contexto do algoritmo EPSO (a solução encontrada em [Souza et al. 2013] aplica apenas a abordagem cooperativa). Durante os experimentos e análise dos resultados obtidos até aquele momento, concluiu-se que a utilização da equação de atualização de velocidade aplicada às partículas originais encontrada no EPSO (Equação 2.22), apesar de demonstrar uma rápida convergência, apresentava dificuldades na obtenção do melhor resultado em relação ao PSO. Isso ocorria devido a estagnação em melhores locais, ocasionada pela baixa variabilidade obtida pelas réplicas nos parâmetros $mw^*_{(i)}$, $mC^*_{1(i)}$, $mC^*_{2(i)}$ e $mC^*_{3(i)}$ (enxame mestre) a partir de um determinado ponto na execução, especialmente nas últimas iterações do processo de busca. Fazia-se necessária a aplicação de um processo de atualização de velocidade das partículas originais que permitisse a variabilidade dos fatores de inércia, interação individual e social, independente do desempenho das EE, as quais atuariam como um processo de busca e otimização auxiliar.

A solução encontrada foi a reinserção da atualização de velocidade do algoritmo PSO clássico, aplicada às partículas originais. Esta pequena alteração se mostrou fundamental para um aumento no desempenho, tanto em relação ao EPSO (melhor resultado), quanto ao PSO (rápida convergência). Com base nisso, desenvolveu-se o algoritmo ***Particle Swarm Optimization with Evolutionary Estrategies*** (PSO+EE) e, a partir dele, o ***Competitive Evolutionary Multi Swarm Optimization*** (CEMSO) [1].

[1] Consultar o Capítulo 5 para mais informações

Durante a publicação de [Souza et al. 2013] no *20st European Conference on Artificial Intelligence* (ECAI-2012), ocorrida em Montpelier (França), diversas conversas ocorreram com o acadêmico Jun Sun (co-criador do algoritmo QPSO), sendo que seu mais recente trabalho ([Sun et al. 2013]) consistia em um algoritmo híbrido que agregava o QPSO com técnicas de programação genética. Deste encontro, surgiu a idéia de utilizar a mesma abordagem encontrada no algoritmo PSO+EE no QPSO.

Após diversos testes e modificações no cálculo de *Learning Inclination Point* aplicado às réplicas, bem como a implementação do mesmo no enxame mestre, foi gerada duas novas metahurísticas híbridas: ***Quantum Particle Swarm Optimization with Evolutionary Estrategies*** (QPSO+EE) e sua versão multi-enxame com topologia mestre-escravos, denominada ***Competitive Quantum-Behaviour Evolutionary Multi Swarm Optimization*** (CQEMSO).

Paralelamente ao desenvolvimento dos algoritmos CEMSO e CQEMSO, foram introduzidas três metaheurísticas multi-enxame baseadas nos algoritmos PSO, EPSO e QPSO. Surgia então os algoritmos denominados ***Classic Competitive Multi Swarm Optimization*** (COMSO, baseada no trabalho de [Niu et al. 2005]), ***Classic Competitive Evolutionary Multi Swarm Optimization*** (COEMSO) e ***Classic Competitive Quantum-Behaviour Multi Swarm Optimization*** (COQMSO). Com essas três soluções, seria possível expandir os algoritmos EPSO e QPSO para uma abordagem multi-enxame de topologia mestre-escravos, obtendo soluções que usufruam das performances dessa categoria de algoritmos.

O último mecanismo desenvolvido e exposto nesta dissertação foram as noções de espaços virtuais aplicados a condições de contorno de espelhamento e equivalência, sendo que ambas estariam operando em conjunto com a técnica de reposicionamento *Damping*. O objetivo deste método seria a geração de três soluções distintas da mesma partícula, onde cada uma daria ênfase a uma abordagem de correção de posição diferente aos valores que excederam o espaço de busca: *Damping* para valores de posição localizados nos limites estabelecidos; *Espelhamento* para valores de posição próximos aos limites no decorrer das iterações; *Equivalência* para valores de posição realocados com base no espaço de busca virtual.

Após a finalização da documentação dos algoritmos multi-enxame expostos, fazia-se necessária a utilização de uma arquietura paralela ou distribuída para a utilização dos mesmos em tempo hábil. Dada o histórico de uso anterior a esta dissertação do paradigma GPGPU ([Souza 2010]), bem como outras características como a facilidade de programação, portabilidade, baixo-custo e capacidade de processamento paralelo massivo, foi selecionada a arquitetura CUDA como ambiente de execução paralela.

Partindo da solução apresentada em [Souza 2010, Souza et al. 2011], foram feitas diversas adaptações que visavam a utilização dos enxames escravos de maneira organizada e mais adequada possível ao hardware. Desse estudo, foram gerados os modelos de organização de *threads* e *blocks* apresentados no Capítulo 5, tópico 5.6, onde se estabeleceu a relação "Um *block*, um enxame", além da já documentada "Uma *thread*, uma partícula". Este modelo pode ser encontrado em [Souza et al. 2013][2].

No intuito de aplicar os algoritmos desenvolvidos em problemas próximos aos encontrados no mundo real e validar seus resultados, foram utilizados quatro problemas de engenharia com vasta utilização na literatura, expressados analiticamente e que envolvem diversas restrições. Os dados obtidos demonstraram que os algoritmos CEMSO e CQEMSO alcançaram resultados superiores em todos os problemas quando comparados a outras soluções encontradas na literatura científica.

É importante ressaltar que os algoritmos PSO+EE e QPSO+EE apresentaram melhorias significativas em relação aos algoritmos PSO, EPSO e QPSO no que diz respeito a média, melhores *fitness* e variância. Consequentemente, isso resulta em um impacto positivo nas suas respectivas versões multi-enxames (CEMSO e CQEMSO)[3].

Com base nos resultados obtidos, conclui-se a priori que os objetivos estipulados para este trabalho foram cumpridos ao propor e desenvolver os algoritmos CEMSO/CQEMSO e, também, foram de certa forma extrapolados[4], com a apresentação das condições de contorno de equivalência e espelhamento de coordenadas, bem como dos algoritmos COMSO, COEMSO e COQMSO. Juntas, todas as soluções desenvolvidas constituem um conjunto poderoso de algoritmos, tanto em termos de desempenho quanto em tempo de execução.

[2]Em relação a publicação escrita pelo autor ([Souza et al. 2013]), esta dissertação teve como melhorias e novas soluções a partir do desenvolvimento da mesma, os seguintes pontos:

1. Modificação da atualização de velocidade para partículas originais, passando a utilizar o procedimento do PSO clássico (PSO+EE);
2. Implementação dos algoritmos COMSO, COEMSO, COQMSO;
3. Introdução da metaheurística QPSO+EE e CQEMSO;
4. Aplicação de três condições de contorno distintas para cada partícula, sendo que duas foram concebidas pelo autor (espelhamento e equivalência).

[3]No caso do problema DPV, o desempenho dos algoritmos CEMSO e CQEMSO em termos de média dos enxames escravos e convergência passam a ser melhores a partir de uma determinada quantidade de iterações, apesar de apresentarem ao final, os melhores resultados. Para maiores informações, ver Capítulo 4, subtópico 4.1.2

[4]O projeto de dissertação inicial não previa as condições de contorno de equivalência e espelhamento de coordenadas, nem os algoritmos COMSO, COEMSO e COQMSO

5.2 Limitações da Dissertação

Apesar dos resultados obtidos no desenvolvimento deste trabalho, há limitações referente ao que foi apresentado. Dentre elas, algumas são destacadas a seguir, bem como possíveis soluções.

- ***Limitação da quantidade de partículas por enxame***: Com a utilização da abordagem "Um *block*, um enxame", a quantidade de partículas em cada população está limitada ao valor máximo de *threads* por *block*. No caso da arquitetura CUDA, as placas mais antigas suportam até 512 *threads* (partículas) por *block* (capacidade de computabilidade entre 1.0 e 1.3) e as mais modernas suportam até 1024 *threads* (partículas) por *block* (capacidade de computabilidade 2.0 ou superior). Uma possível solução seria a utilização de uma cardinalidade diferente para a relação *threads*-partícula, onde uma *thread* se encarregaria de duas ou mais partículas entretanto, tal abordagem ainda não foi testada;
- ***Quantidade de problemas diminuída e ausência de problemas não-restritivos***: Inicialmente foi proposto um conjunto muito maior de problemas de otimização para serem utilizados nas simulações, entretanto, devida a quantidade tempo necessária para codificar esses problemas, especialmente àquelas relacionadas as alterações do código-fonte para a inserção de problemas não-restritivos, houve uma revisão nesse conjunto. Inicialmente seriam 8 problemas não-restritivos e 7 de engenharia (restritivos). Para a implementação desses problemas em tempo hábil, seria necessária a escrita de um código que facilite a inserção de novos problemas, o que, a princípio, demandaria alterações significativas;
- ***Ausência da abordagem cooperativa***: Durante os experimentos realizados no desenvolvimento desta dissertação, foi constatado que a abordagem cooperativa, embora apresentasse resultados relevantes, obtinha valores de média e variância inferiores aos algoritmos expostos nesta dissertação. Embora sejam apresentados no Capítulo 5, esta abordagem ainda está sendo aprimorada e os resultados serão divulgados em um trabalho posterior;
- ***Ausência de um modificador de direção do vetor velocidade nas condições de contorno de espelhamento***: Apesar do procedimento de espelhamento de coordenadas auxiliar na exploração de regiões próximas dos limites nas útimas iterações, não foi desenvolvido um mecanismo responsável pela configuração da direção do vetor de velocidade, ficando assim dependente da influênica exercida pelos componentes de inércia, interação social e individual.

5.3 Trabalhos Futuros

Ao considerar as possibilidades de expansão do conteúdo desenvolvido, vislumbra-se a realização de trabalhos futuros. Além de buscar soluções às limitações citadas no tópico 5.2, procura-se ampliar o uso dos algoritmos desenvolvidos. Sendo assim, a seguir estão listados alguns trabalhos futuros.

- ***Melhorias nos algoritmos multi-enxame sob a arquitetura CUDA***: Os gráficos relacionados ao tempo de execução indicam que, apesar dos algoritmos apresentarem resultados relevantes (entre 0.003 até 1.7 segundos), observa-se um aumento exponencial do tempo na medida em que a quantidade de iterações aumenta. Dentre os aprimoramentos voltados para a redução do tempo, propõe-se a utilização de *Shared Memory* para a manipulação dos dados alocados na memória da GPU, re-organização das estruturas de dados geradas com o objetivo de aprimorar a coalescência de memória e um estudo detalhado da complexidade do algoritmo;
- ***Portabilização dos algoritmos COMSO, COEMSO, COQMSO, CEMSO e CQEMSO para outras plataformas paralelas***: Considerando o uso bem sucedido da arquitetura CUDA na implementação dos algoritmos expostos, o próximo passo consiste em um levantamento técnico acerca de outras plataformas de computação paralela e/ou distribuídas, bem como a viabilidade (e.g. custo-benefício, aplicações práticas). Estuda-se também a possibilidade do uso da arquitetura CUDA com outras soluções (e.g. MPI, PVM), bem como uma implementação sob a arquitetura FPGA (*Field Programmable Gate Array*);
- ***Novas publicações***: Realizar publicações em periódicos classificados como Qualis A2 e A1, conferências, capítulos de livro, etc;
- ***Aplicação dos algoritmos desenvolvidos na solução de problemas relacionados a projetos acadêmicos e industriais***: Esta etapa consiste no levantamento de projetos onde seja possível o estabelecimento de parcerias visando a aplicação dos algoritmos desenvolvidos as respectivas necessidades;
- ***Aplicação da estrutura paralela desenvolvida em outras metaheurísticas sob a arquitetura CUDA***: Em parceria com alunos do Laboratório de Computação Natural (CESUPA) e do Programa de Pós-Graduação em Ciência da Computação (PPGCC-UFPA), esse ponto consiste em utilizar as estruturas encontradas nos algoritmos multi-enxame desenvolvidas neste trabalho e aplicá-las em outras metaheurísticas (e.g. Algoritmos Genéticos (GA), Sistemas Imunológicos Artificiais (AIS), Otimização por Colônia de Formigas (ACO));

- ***Inserir os fundamentos da Teoria dos Jogos*[5] *sob o contexto dos algoritmos PSO+EE, QPSO+EE, CEMSO e CQEMSO***: Essa fase consiste no estudo e desenvolvimento de uma nova metaheurística híbrida de otimização, a qual introduz os processo relacionados ao que [Teixeira 2012] refere como Interação Social[6] na estrutura dos enxames de partículas com EE e aplicando-se os conceitos da Teoria dos Jogos, tendo como base os *insigts* e mecanismos encontrados nos algoritmos desenvolvidos por [Teixeira 2012, Teixeira et al. 2010]. O objetivo desta nova solução será aprimorar o processo de EE através do uso da Teoria dos Jogos. Desde outubro de 2013, o autor desta dissertação tem desenvolvido em parceria com o mestrando Walter Avelino e Prof. Dr. Otávio Noura Teixeira, uma versão do PSO+EE que inclui mecanismos de seleção baseados na Teoria dos Jogos, bemo como a portabilização dos algoritmos desenvolvidos em [Teixeira 2012] sob a arquitetura CUDA utilizando a mesma organização de *threads* e *blocks* encontradas nos algoritmos COMOS, COEMSO, COQMSO, CEMSO e CQEMSO.

[5]Baseando-se nos trabalhos desenvolvidos por [Teixeira 2012, Teixeira et al. 2010]

[6]É necessário fazer a distinção entre o termo interação social empregado nos algoritmos de otimização por enxame de partículas e o conceito usado por [Teixeira 2012], cuja definição pode ser encontrada abaixo:

> *O termo Interação Social representa muito bem o que se deseja com esse processo, pois significa a interação entre indivíduos de uma populacção e não somente o encontro de dois indivíduos para reproduzirem, conforme ocorre na teoria dos Algoritmos Genéticos. Deve-se ter em mente que ele está diretamente ligado a quaisquer situações de conflito de interesses que possam vir a existir entre os indivíduos de uma população, o que pode ser demonstrado por um jogo social, econômico ou até mesmo evolutivo.*

Capítulo 6

Algoritmos COMSO (PSO), COEMSO (EPSO) e COQMSO (QPSO)

6.1 Introdução

Esta seção expõe o fluxograma e pseudocódigo dos algoritmos multi-enxame de topologia mestre-escravos baseados nas metaheurísticas PSO, EPSO e QPSO. A implementação destes algoritmos tem por objetivo os seguintes pontos:

- ***Comparação de desempenho***: Os algoritmos COMSO, COEMSO e COQMSO são utilizados para a comparação dos resultados obtidos (i.e. média, melhor *fitness* e variância) com os algoritmos CEMSO e CQEMSO, bem como as versões mono-enxame dos mesmos (i.e. PSO, PSO+EE, EPSO, EPSO+EE, QPSO, QPSO+EE);
- ***Comparação do tempo de execução***: Os algoritmos COMSO, COEMSO e COQMSO são utilizados para a comparação do tempo de execução com os algoritmos CEMSO e CQEMSO, bem como as versões mono-enxame dos mesmos (i.e. PSO, PSO+EE, EPSO, EPSO+EE, QPSO, QPSO+EE);
- ***Implementação da solução baseada no algoritmo descrito em [Niu et al. 2005] sob a arquitetura CUDA (COMSO)***: O algoritmo COMSO consiste em uma implementação ulti-enxame do PSO sob topologia mestre-escravos em CUDA;
- ***Implementação dos algoritmos EPSO e QPSO sob um ambiente multi-enxame com topologia mestre-escravos em CUDA (COEMSO e COQMSO)***: Com o levantamento bibliográfico realizado para o desenvolvimento deste trabalho, não foram encontradas implementações dos algoritmos EPSO e QPSO utilizando a abordagem de [Niu et al. 2005][1].

[1]Levantamento bibliográfico ocorrido entre 2011 até 2013, com o fechamento deste documento

6.2 Pseudocódigo e Fluxograma (Algoritmo COMSO)

O algoritmo 18 mostra a inicialização dos dados nos enxames escravos do algoritmo COMSO sob a arquitetura CUDA.

Algoritmo 18: COMSO - Inicialização dos enxames escravos - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de velocidade;
                Inicializar os valores de posição;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 19 mostra a inicialização dos dados no enxame mestre do algoritmo COMSO sob a arquitetura CUDA.

Algoritmo 19: COMSO - Inicialização do enxame mestre - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de velocidade;
                Inicializar os valores de posição;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g^M);
        end
    end
end
```

O algoritmo 20 mostra o processo de busca e otimização nos enxames escravos do algoritmo COMSO sob a arquitetura CUDA.

Algoritmo 20: COMSO com fator de inércia (enxames escravos) - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a partícula original (Equação 2.5);
                Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
                Atualizar valor de posição da variável j para a partícula original (Equação 2.6);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            Obter os melhor local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 21 mostra o processo de busca e otimização no enxame mestre do algoritmo COMSO sob a arquitetura CUDA.

Algoritmo 21: COMSO com fator de inércia (enxame mestre) - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a partícula original (Equação 2.37 ou
                2.39);
                Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
                Atualizar valor de posição da variável j para a partícula original (Equação 2.6);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            Obter os melhor local da partícula (P_i^M);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g^M);
                if P_g^S for melhor que P_g^M then
                    Atribuir o valor de P_g^S a P_g^M;
        end
    end
end
```

O Algoritmo 22 mostra a execução de COMSO na íntegra sob a arquitetura CUDA.

Algoritmo 22: Algoritmo COMSO em CUDA

Alocar memória para as estruturas dos enxames escravos (posição (X), velocidade (V)) (GPU);
Alocar memória para o melhores valores dos enxames escravos (local (P_i), global (P_g)) (GPU);
Alocar memória para o melhor resultado obtido pelos enxames escravos (P_g^S) (CPU e GPU);
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização dos enxames escravos (Algoritmo 18);
Sincronizar GPU;
Inicializar o melhor resultado encontrado pelos enxames escravos (P_g^S);
Alocar memória para estruturas do enxame mestre (posição (X), velocidade (V)) (GPU);
Alocar memória para melhores valores dos enxame mestre (local (P_i), global (P_g^M)) (GPU);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização do enxame mestre (Algoritmo 19);
Sincronizar GPU;
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
for $iter \leftarrow 1$ **to** *(QUANTIDADE DE ITERAÇÕES*-1 **do**
    Atualizar fator de inércia (Equação 2.1 ou 2.2);
    Executar o *kernel* de busca e otimização enxames escravos (Algoritmo 20);
    Sincronizar GPU;
    Obter o melhor resultado encontrado pelos enxames escravos (P_g^S);
    Executar o *kernel* de busca e otimização enxame mestre (Algoritmo 21);
    Sincronizar GPU;
Copiar os valores de melhor global dos enxames escravos para a memória principal (GPU para CPU);
Copiar os valores de melhor global do enxame mestre para a memória principal (GPU para CPU);
Liberar memória;

6.3 Pseudocódigo e Fluxograma (Algoritmo COEMSO)

O algoritmo 23 mostra a inicialização dos dados nos enxames escravos do algoritmo COEMSO sob a arquitetura CUDA.

Algoritmo 23: COEMSO - Inicialização dos enxames escravos - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de velocidade;
                Inicializar os valores de posição;
            Inicializar os valores de fator de inércia (w_(i)) para os enxames escravos;
            Inicializar os valores de interação individual (C_1(i)) para os enxames escravos;
            Inicializar os valores de interação social (C_2(i)) para os enxames escravos;
            Inicializar os valores para o fator de pertubação do melhor global (ω_i) para os enxames
            escravos;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 24 mostra a inicialização dos dados no enxame mestre do algoritmo COEMSO sob a arquitetura CUDA.

Algoritmo 24: COEMSO - Inicialização do enxame mestre - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de velocidade;
                Inicializar os valores de posição;
            Inicializar os valores de fator de inércia (w_(i)) para o enxame mestre;
            Inicializar os valores de interação individual (C_1(i)) para o enxame mestre;
            Inicializar os valores de interação social (C_2(i)) para o enxame mestre;
            Inicializar os valores de interação social dos enxames escravos (C_3(i)) para o enxame mestre;
            Inicializar os valores para o fator de pertubação do melhor global (ω_i) e atribuir ao enxame
            mestre;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i^M);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g^M);
        end
    end
end
```

O algoritmo 25 mostra procedimento de busca e otimização envolvendo as partículas originais e o uso de estratégias evolutivas para a geração das réplicas nos enxames escravos do algoritmo COEMSO sob a arquitetura CUDA.

Algoritmo 25: COEMSO com fator de inércia (enxames escravos) - CUDA

Iniciar o *Kernel* dos Enxames Escravos
  Executar em Paralelo Todos os Enxames Escravos (multi-*Block*)
    Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-*Thread*)
    $i \leftarrow$ **índice da *thread***
      for $j \leftarrow 0$ **to** *(QUANTIDADE DE VARIÁVEIS*-1*)* **do**
        Atualizar valor de velocidade da variável j para a partícula original (Equação 2.22);
        Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
        Atualizar valor de posição da variável j para a partícula original (Equação 2.6 ou 2.24);
        Aplicar condições de contorno descrita no Algoritmo 4;
      if *Houver cópias da partícula com resultados de condições de contorno distintas* **then**
        Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
      Avaliar partícula (*Fitness*);
      for $k \leftarrow 0$ **to** *(QUANTIDADE DE RÉPLICAS*-1*)* **do**
        Gerar novo valor para o fator de inércia atravé da mutação (2.17));
        Gerar novo valor para o fator de interação individual através da mutação (2.18));
        Gerar novo valor para o fator de interação social através da mutação (2.19));
        Gerar novo valor para o fator de pertubação atravé da mutação (2.20));
        for $j \leftarrow 0$ **to** *(QUANTIDADE DE VARIÁVEIS*-1*)* **do**
          Atualizar valor de velocidade da variável j para a réplica k (Equação 2.23);
          Aplicar ajuste de velocidade para a réplica k (Equação 2.9);
          Atualizar valor de posição da variável j para a réplica k (Equação 2.25);
          Aplicar condições de contorno descrita no Algoritmo 5;
        if *Houver cópias da partícula com resultados de condições de contorno distintas* **then**
          Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
        Avaliar réplica (*Fitness*);
        Comparar réplica e partícula original;
        if *fitness(Replica) melhor do que fitness(Original)* **then**
          Substituir partícula original pela réplica (*Original* = *Replica*);
          Substituir fatores $C_{1(i)}$, $C_{2(i)}$ e $w_{(i)}$ por $mC^*_{1(i)}$, $mC^*_{2(i)}$ e $mw^*_{(i)}$;
          Substituir *fitness* original por *fitness* réplica;
      Obter os melhor local da partícula (P_i);
      Sincronizar as *threads* do respectivo *block*;
      if *Índice da thread de cada block (threadIdx) for igual a zero* **then**
        Obter o melhor global de cada enxame escravo (P_g);
    end
  end
end

O algoritmo 26 mostra o procedimento de busca e otimização envolvendo as partículas originais e o uso de estratégias evolutivas para a geração das réplicas no enxame mestre do algoritmo COEMSO sob a arquitetura CUDA.

Algoritmo 26: COEMSO com fator de inércia (enxame mestre) - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de velocidade da variável j para a partícula original (Figura 3.5);
                Aplicar ajuste de velocidade para a partícula original (Equação 2.9);
                Atualizar valor de posição da variável j para a partícula original (Equação 2.6);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            for k ← 0 to (QUANTIDADE DE RÉPLICAS−1) do
                Gerar novo valor para o fator de inércia atravé da mutação (2.17));
                Gerar novo valor para o fator de interação individual através da mutação (2.18));
                Gerar novo valor para o fator de interação social através da mutação (2.19));
                Gerar novo valor para o fator de interação social dos enxames escravos através da
                mutação (3.4));
                Gerar novo valor para o fator de pertubação para o enxame mestre atravé da
                mutação (2.20));
                for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                    Atualizar valor de velocidade da variável j para a réplica k (Figura 3.6);
                    Aplicar ajuste de velocidade para a réplica k (Equação 2.9);
                    Atualizar valor de posição da variável j para a réplica k (Equação 2.25);
                    Aplicar condições de contorno descrita no Algoritmo 5;
                if Houver cópias da partícula com resultados de condições de contorno distintas
                then
                    Selecionar a melhor cópia da partícula e classifica-la como a solução da
                    mesma;
                Avaliar réplica (Fitness);
                Comparar réplica e partícula original;
                if fitness(Replica) melhor do que fitness(Original) then
                    Substituir partícula original pela réplica (Original = Replica);
                    Substituir fatores C1(i), C2(i) e w(i) por mC*1(i), mC*2(i) e mw*(i);
                    Substituir fitness original por fitness réplica;
            Obter os melhor local da partícula (P^M_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P^M_g);
                if P^S_g for melhor que P^M_g then
                    Atribuir o valor de P^S_g a P^M_g;
        end
    end
end
```

O Algoritmo 27 mostra o execução de COEMSO na íntegra sob a arquitetura CUDA.

Algoritmo 27: Algoritmo COEMSO completo - CUDA

Alocar memória para as estruturas dos enxames escravos (posição (X), velocidade (V)) (GPU);
Alocar memória para o melhores valores dos enxames escravos (local (P_i), global (P_g)) (GPU);
Alocar memória para o melhor resultado obtido pelos enxames escravos (P_g^S) (CPU e GPU);
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização dos enxames escravos (Algoritmo 23);
Sincronizar GPU;
Inicializar o melhor resultado encontrado pelos enxames escravos (P_g^S);
Alocar memória para estruturas do enxame mestre (posição (X), velocidade (V)) (GPU);
Alocar memória para melhores valores dos enxame mestre (local (P_i), global (P_g^M)) (GPU);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização do enxame mestre (Algoritmo 24);
Sincronizar GPU;
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
for $iter \leftarrow 1$ **to** *(QUANTIDADE DE ITERAÇÕES*-1 **do**
    Executar o *kernel* de busca e otimização enxames escravos (Algoritmo 25);
    Sincronizar GPU;
    Obter o melhor resultado encontrado pelos enxames escravos (P_g^S);
    Executar o *kernel* de busca e otimização enxame mestre (Algoritmo 26);
    Sincronizar GPU;
Copiar os valores de melhor global dos enxames escravos para a memória principal (GPU para CPU);
Copiar os valores de melhor global do enxame mestre para a memória principal (GPU para CPU);
Liberar memória;

6.4 Pseudocódigo e Fluxograma (Algoritmo COQMSO)

O algoritmo 28 mostra a inicialização dos dados nos enxames escravos do algoritmo COQMSO sob a arquitetura CUDA.

Algoritmo 28: COQMSO - Inicialização dos enxames escravos - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos
        (multi-Thread) i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de posição;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 29 mostra a inicialização dos dados no enxame mestre do algoritmo COQMSO sob a arquitetura CUDA.

Algoritmo 29: COQMSO - Inicialização do enxame mestre - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre
        (multi-Thread) i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Inicializar os valores de posição;
            Avaliar partícula (Fitness);
            Inicializar os melhores local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g^M);
        end
    end
end
```

O algoritmo 30 mostra o processo de busca e otimização nos enxames escravos do algoritmo COQMSO sob a arquitetura CUDA.

Algoritmo 30: COQMSO com fator de inércia (enxames escravos) - CUDA

```
Iniciar o Kernel dos Enxames Escravos
    Executar em Paralelo Todos os Enxames Escravos (multi-Block)
        Executar em Paralelo Todas as Partículas dos Enxames Escravos (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de ponto de inclinação (LIP ou p) (Equação 2.28);
                Atualizar valor de posição da variável j (Equação 2.36);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            Obter os melhor local da partícula (P_i);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global de cada enxame escravo (P_g);
        end
    end
end
```

O algoritmo 31 mostra o processo de busca e otimização no enxame mestre do algoritmo COQMSO sob a arquitetura CUDA.

Algoritmo 31: COQMSO com fator de inércia (enxame mestre) - CUDA

```
Iniciar o Kernel do Enxames Mestre
    Executar em Paralelo o Enxame Mestre (single-Block)
        Executar em Paralelo Todas as Partículas do Enxame Mestre (multi-Thread)
        i ← índice da thread
            for j ← 0 to (QUANTIDADE DE VARIÁVEIS−1) do
                Atualizar valor de ponto de inclinação (LIP ou p) para o enxame mestre (Equação
                3.30);
                Atualizar valor de posição da variável j (Equação 3.31);
                Aplicar condições de contorno descrita no Algoritmo 4;
            if Houver cópias da partícula com resultados de condições de contorno distintas then
                Selecionar a melhor cópia da partícula e classifica-la como a solução da mesma;
            Avaliar partícula (Fitness);
            Obter os melhor local da partícula (P_i^M);
            Sincronizar as threads do respectivo block;
            if Índice da thread de cada block (threadIdx) for igual a zero then
                Obter o melhor global do enxame mestre (P_g^M);
                if P_g^S for melhor que P_g^M then
                    Atribuir o valor de P_g^S a P_g^M;
        end
    end
end
```

O Algoritmo 32 mostra a execução de COQMSO na íntegra sob a arquitetura CUDA.

Algoritmo 32: Algoritmo COQMSO completo - CUDA

Alocar memória para as estruturas dos enxames escravos (posição (X), velocidade (V)) (GPU);
Alocar memória para o melhores valores dos enxames escravos (local (P_i), global (P_g)) (GPU);
Alocar memória para o melhor resultado obtido pelos enxames escravos (P_g^S) (CPU e GPU);
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização dos enxames escravos (Algoritmo 28);
Sincronizar GPU;
Inicializar o melhor resultado encontrado pelos enxames escravos (P_g^S);
Alocar memória para estruturas do enxame mestre (posição (X), velocidade (V)) (GPU);
Alocar memória para melhores valores dos enxame mestre (local (P_i), global (P_g^M)) (GPU);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
Executar o *kernel* de inicialização do enxame mestre (Algoritmo 29);
Sincronizar GPU;
Definir a quantidade de *blocks* para os enxames escravos na GPU (um *block*, um enxame escravo);
Definir a quantidade de *threads* para os enxames escravos na GPU (uma *thread*, uma partícula);
Definir a quantidade de *blocks* para o enxame mestre na GPU para 1 (um *block*, um enxame mestre);
Definir a quantidade de *threads* para o enxame mestre na GPU (uma *thread*, uma partícula);
for $iter \leftarrow 1$ **to** *(QUANTIDADE DE ITERAÇÕES*-1 **do**
    Atualizar fator β (Equação 2.32 ou 2.33);
    Atualizar fator α (Equação 2.29 ou 2.30);
    Obter a média dos melhores locais dos enxames escravos (Equação 2.31);
    Executar o *kernel* de busca e otimização enxames escravos (Algoritmo 30);
    Sincronizar GPU;
    Obter o melhor resultado encontrado pelos enxames escravos (P_g^S);
    Obter a média dos melhores locais do enxame mestre (Equação 2.31);
    Executar o *kernel* de busca e otimização enxame mestre (Algoritmo 31);
    Sincronizar GPU;
Copiar os valores de melhor global dos enxames escravos para a memória principal (GPU para CPU);
Copiar os valores de melhor global do enxame mestre para a memória principal (GPU para CPU);
Liberar memória;

Referências Bibliográficas

Aarts, E. & J. Korst (1989), *Simulated Annealing and Boltzman Machines - A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley and Sons Inc.

Arora, J. (2004), *Introduction to Optimum Design*, 2ª edição, Academic Press.

Back, T., D. B. Fogel & Z. Michalewicz (2000), 'Evolutionary computation 1 basic algorithms and operators', *Institute of Physiscs Publishing (IOP)* .

Belegundu, A. & J. Arora (1985), 'A study of mathematical programming methods for structural optimization. part i: Theory', *International Journal for Numerical Methods in Engineering* **21(9)**, 1583–1599.

Bonabeau, E., M. Dorigo & T. Theraulaz (1999), *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press.

Brajevic, I., M. Tuba & M. Subotic (2010), 'Improved artificial bee colony algorithm for constrained problems', *Proceedings of the 11th WSEAS International Conference on Neural Networks, Fuzzy Systems and Evolutionary Computing* pp. 185–190.

Cagnina, L., S. Esquivel & C. Coello Coello (2008), 'Solving engineering optimization problems with the simple constrained particle swarm optimizer', *Informatica,* **32(3)**, 319–326.

Coello, C. Coello & E. Montes (2002), 'Constraint-handling in genetic algorithms through the use of dominance-based tournament selection', *Advanced Engineering Informatics* pp. 193–203.

Costa, G. R. (2013), Comparação de sequências biológicas utilizando computação heterogênea com openCL, Trabalho de conclusão de curso, Universidade de Brasília (UnB).

CST, Computer Simulation Technology (2013), 'GPU computing - CUDA and CST', `http://www.cst.com/content/products/mws/{GPU}.aspx`.

Darwin, C. (1859), *Sobre a Origem das Espécies por Meio da Seleção Natural ou a Preservação de Raças Favorecidas na Luta pela Vida*, John Murray.

Dasgupta, D. (1999), *Artificial Immune Systems and Their Applications*, Springer-Verlag.

de Castro, L. N. (2010), *Computação Natural: Uma Jornada Ilustrada.*, Editora Livraria da Física.

de Castro, L. N. & J. I. Timmis (2002), *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag.

den Bergh, H. Van & A. P. Engelbrecht (2004), 'A cooperative approach to particle swarm optimization', *IEEE Transactions on Evolutionary Computation* pp. 225–239.

Dorigo, M. & L. M. Gambardella (1997), 'Ant colony system: A cooperative learning approach to the traveling salesman problem', *IEEE Transactions on Evolutionary Computation* **1(1)**, 53–66.

Dorigo, M., V. Maniezzo & A. Colorni (1996), 'The ant system: Optimization by a colony of cooperating agents', *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics* **26(1)**(29-41).

Eberhart, R. & P. Simpson (1996), *Computational Intelligence PC Tools*, 1ª edição, Academic Press.

Eberhart, R. & Y. Shi (1998), A modified particle swarm optimizer, IEEE, IEEE Press, pp. 69–73.

Eberhart, R. & Y. Shi (2000), 'Comparing inertia weights and constriction factors', *Proceedings of the Congress on Evolutionary Computing* **Particle Swarm Optimization**, 84–89.

Eberhart, R. & Y. Shi (2001), Tracking and optimizing dynamic systems with particle swarms, *em* 'Proceedings of IEEE Congress on Evolutionary Computation', pp. 94–97.

El-Abd, M. & M. Kamel (2008), 'A taxonomy of cooperative particle swarm optimizers', *International Journal of Computational Intelligence Research* pp. 137–144.

Engelbrecht, A. P. (2007), *Computational Intelligence: An Introduction*, John Wiley and Sons Inc.

Filho, C. J. A. Bastos, M. P. Caraciolo, P. B. Miranda & D. F. Carvalho (2008), Multi ring PSO, *em* 'Anais do SBRN 2008', SBRN'2008 (the 10th Brazilian Symposium on Neural Networks).

Forbes, N. (2005), *Imitation of Life: How Biology is Inspiring Computing*, MIT Press.

Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company.

Hastings, W. K. (1970), 'Monte carlo sampling methods using markov chains and their applications', *Biometrika* **57**(1), 97–109.

Haupt, R. L. & S. E. Haupt (1998), *Practical Genetic Algoritms*, 1ª edição, John Wiley and Sons Inc.

Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, Prentice Hall Press.

He, Q. & L. Wang (2007), 'An effective co-evolutionary particle swarm optimization for constrained engineering design problems', *Engineering Applications of Artificial Intelligence* pp. 89–99.

Herrera, B. A. L. M. (2007), Combinação de enxame de partículas com inspiração quântica e método lin-kernighan-helsgaun aplicada ao problema do caixeiro viajante, Dissertação de mestrado, Pontifícia Universidade Católica do Paraná (PUC-PR).

Hsuan-Ming, F. (2005), 'Particle swarm optimization learning fuzzy systems design', *Third International Conference on Information Technology and Applications (ICITA)* **1**, 363–366.

Hwu, W. W., K. Keutzer & K. Mattson (2008), 'The concurrency challenging', *IEEE Design and Test of Computers* pp. 312–320.

IEEE (2007), 'Ieee 754: Standard for binary floating-point arithmetic', `http://grouper.ieee.org/groups/754/`.

Kennedy, J. & R. Eberhart (1995), Particle swarm optimization, *em* 'Proceedings of the IEEE Int. Conf. on Neural Networks', IEEE, IEEE Press, pp. 1942–1948.

Kennedy, J. & R. Eberhart (2001), *Swarm Intelligence*, 2ª edição, Morgan Kaufmann.

Kirk, D. B. & W. W. Hwu (2010), *Programming Massively Parallel Processors: A Hands-on Approach*, 1ª edição, Elsevier and Morgan Kaufman.

Kuok, K. K., S. Harum & S. M. Shamsuddin (2010), 'Particle swarm optimization feed-forward neural network for modeling runoff', *International Journal of Environment, Science and Technology* pp. 67–78.

Leite, H., J. Barros & V. Miranda (2010), The evolutionary algorithm EPSO to coordinate directional overcurrent relays, *em* '10th IET International Conference on Developments in Power System Protection (DPSP 2010). Managing the Change', IEEE Press, pp. 1–5.

Linden, R. (2008), *Algoritmos Genéticos*, 2ª edição, Editora Braspot.

Liu, H., S. Xu & X. Liang (2008), 'A modified quantum-behaved particle swarm optimization for constrained optimization', *ï¿ ¼International Symposium on Intelligent Information Technology Application Workshop* .

Liu, T. C. (2006), Developing a Fuzzy Proportional-Derivative Controller Optimization Engine for Engineering Optimization Problems, Tese de doutorado, Universidade Chungli - Taiwan.

Mani, A. & C. Patvardhan (2010), 'An adaptive quantum evolutionary algorithm for engineering optimization problems', *International Journal of Computer Applications* **1**(22), 43–48.

Mathworks (2013), 'Matlab GPU computing support for NVIDIA cuda-enabled GPUs', `http://www.mathworks.com/discovery/matlab-{GPU}.html`.

Medeiros, T. H., L. F. W. Góes, M. B. Carvalho & I. Menezes (2005), *Computação Bioinspirada Aplicada à Robótica*, Anais da Simpósio da Escola Regional de Informática de Minas Gerais.

Mikki, S. M. & A. A. Kishk (2008), *Particle Swarm Optimization: A Physics-Based Approach*, Morgan and Claypool Publishers.

Miranda, V., H. Keko & A. J. Duque (2007), 'Stochastic star communication topology in evolutionary particle swarm optimization(EPSO)', *IJCIR - International Journal of Computational Intelligence Research* **4**(2).

Miranda, V. & N. Fonseca (2002), EPSO - evolutionary particle swarm optimization, a new algorithm with applications in power systems, *em* 'Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES', Vol. 2, IEEE Press, pp. 745–750.

Mitchell, M. (1999), *An Introduction to Genetic Algorithms*, 1ª edição, MIT Press.

Mori, H. & K. Okawa (2009), 'Integration of parallel epso and variable ts for unit commitment with nonsmooth fuel cost functions', *15th International Conference on Intelligent System Applications to Power Systems (ISAP 09)* pp. 1–6.

Mussi, L., Y. S. G. Nashed & S. Cagnoni (2011), 'GPU-based asynchronous particle swarm optimization', *GECCO 11 Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* **2**, 1555–1562.

Naing, O. W. (2008), A comparison study on particle swarm and evolutionary particle swarm optimization using capacitor placement problem, *em* '2nd IEEE International Conference on Power and Energy (PECon 08)'.

Niu, B., Y. Zhu & X. He (2005), 'Multi-population cooperative particle swarm optimization', *Proceedings of the European Conference on Artificial Life (ECAL 2005)* pp. 874–883.

Niu, B., Y. Zhu, X. He & H. Wu (2007), 'MCPSO: A multi-swarm cooperative particle swarm optimizer', *Applied Mathematics and Computation* pp. 1050–1062.

NVIDIA (2008*a*), 'NVIDIA GeForce 8800 GTX', `http://www.nivida.com/page/geforce_8800.html`.

NVIDIA (2008*b*), Technical brief: NVIDIA geforce GTX 200 GPU architectural overview, Relatório técnico, NVIDIA.

NVIDIA (2012*a*), 'NVIDIA CUDA C programming guide', `http://developer.download.{NVIDIA}.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`.

NVIDIA (2012*b*), NVIDIA's next generation CUDA compute architecture: Kepler gk110, Relatório técnico, NVIDIA Corportation.

Pang, P. (2005), *Quantum Mechanics in Nonlinear Systems*, World Scientific Publishing Company.

Parsopoulos, K. & M. N. Vrahatis (2010), *Particle Swarm Optimization and Intelligence: Advances and Applications*, 1ª edição, IGI Global.

Petr, P., J. Jaros & J. Schwarz (2010), 'Parallel genetic algorithm on the CUDA architecture', *Applications of Evolutionary Computation* pp. 442–451.

Pomeroy, P. (2003), 'An introduction to particle swarm optimization', `http://www.adaptiveview.com/articles/i{PSO}prnt.html`.

Rechenberg, I. (1973), 'Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution', *Frommann-Holzboog Verlag* .

Reynolds, R. & B. Peng (2005), 'Cultural algorithms: Computational modeling of how cultures learn to solve problems: an engineering example', *Cybernetics and Systems: An International Journal* **36**(8), 753–771.

Sanders, J. & E. Kandrot (2010), *CUDA By Example: An Introduction to General-Purpose GPU Programming*, 1ª edição, Addison-Wesley Professional.

Schrodinger, E. (1935), 'Die gegenwartige situation in der quantenmechanik (trad: A situação atual da mecânica quântica)', *Naturwissenschaften* **23**(807), 823–844.

Schwefel, H. P. (1965), Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik, Tese de doutorado, Universidade de Berlim.

Schwefel, H. P. & G. Rudolph (1995), 'Contemporary evolution strategies', *Advances in Artificial Life* pp. 893–907.

Shi, Y. & R. Eberhart (1999), 'Empirical study of particle swarm optimization', *Proceedings Of Congress on Evolutionary Computation* pp. 1945–1950.

Silva, D. J. A., O. N. Teixeira & R. C. L. de Oliveira (2012), Performance study of cultural algorithms based on genetic algorithm with single and multi population for the mkp, *em* 'Bio-Inspired Computational Algorithms and Their Applications', INTECH, capítulo 20, pp. 385–404.

Solomon, S., P. Thulasiraman & R. Thulasiram (2011), 'Collaborative multi-swarm PSO for task matching using graphics processing units', *Proceeding on the GECCO 2011, 13th Annual Conference on Genetic and Evolutionary Computation* **2**, 1563–1570.

Souza, D. L. (2010), PSO-GPU: Implementação da metaheurística de otimização por enxame de partículas (PSO) paralela sob a arquitetura CUDA, Trabalho de conclusão de curso, Centro Universitário do Pará (CESUPA).

Souza, D. L., G. D. Monteiro, T. C. Martins, O. N. Teixeira & V. A. Dmitriev (2011), 'PSO-GPU: Accelerating particle swarm optimization in cuda-based graphics processing units', *GECCO 11 Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* .

Souza, D. L., O. N. Teixeira, D. C. Monteiro & R. C. L. de Oliveira (2013), A new cooperative evolutionary multi-swarm optimizer algorithm based on CUDA architecture applied to engineering optimization, *em* I.Hatzilygeroudis & V.Palade, eds., 'Combinations of Intelligent Methods and Applications', Vol. 23, Springer-Verlag, pp. 95–115.

Souza, D. L., O. N. Teixeira, D. C. Monteiro, R. C. L. de Oliveira & M. A. F. Mollinetti (2014), *A Novel Competitive Quantum-Behaviour Evolutionary Multi-Swarm Optimizer Algorithm Based on CUDA Architecture Applied to Constrained Engineering Design*, Springer-Veerlag.

Stone, J. E., D. Gohara & G. Shi (2010), 'OpenCL: A parallel programming standard for heterogeneous computing systems', *Computing in Science Engineering* **12**(3), 66–73.

Stringhini, D., R. A. Gonçalves & A. Goldman (2012), *Introdução à Computação Heterogênea*, Jornada de Atualização de Informática (JAI) da Sociedade Brasileira de Computação (SBC), capítulo 7.

Sun, J., B. Feng & W. Xu (2004*a*), 'A global search strategy of quantum-behaved particle swarm optimization', *Proceedings of IEEE Congress on Cybernetics and Intelligent Systems* pp. 111–116.

Sun, J., B. Feng & W. Xu (2004*b*), Particle swarm optimization with particles having quantum behavior, *em* 'Congress on Evolutionary Computation (CEC 2004)', Vol. 1, pp. 325–331.

Sun, J., B. Feng & W. Xu (2005), 'Adaptive parameter control for quantum-behaved particle swarm optimization on individual level', *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* pp. 3049–3054.

Sun, J., V. Palade & X. Wu (2013), *Hybrid Approach of Genetic Programming and Quantum-Behaved Particle Swarm Optimization for Modeling and Optimization of Fermentation Processes*, Vol. 23, Springer-Verlag, pp. 117–136.

Teixeira, O. N. (2012), Algoritmo Genético com Interação Social Nebulosa, Tese de doutorado, Universidade Federal do Pará.

Teixeira, O. N., W. A. L. Lobato, H. S. Yanaguibashi, R. V. Cavalcante, D. J. A. Silva & R. C. L. de Oliveira (2011), *Algoritmo Genético com Interação Social na Resolução de Problemas de Otimização Global com Restrições*, 1ª edição, Editora OMNIPAX, capítulo 10, pp. 197–223.

Teixeira, O.N., W. A. L. Lobato, C. T. K. Yasojima, F. H. de Brito, A. N. Teixeira & R. C. L. de Oliveira (2010), 'A new hybrid nature-inspired metaheuristic for problem solving based on the social interaction genetic algorithm employing fuzzy systems', *Proceedings of the* 10^{th} *International Conference on Hybrid Intelligent Systems* pp. 31–36.

Veronese, L. P. & R. A. Krohling (2009), Swarm's flight: Accelerating the particles using C-CUDA., *em* 'CEC 2009: IEEE Congress on Evolutionary Computation', pp. 3264–3270.

Vignati, A. L., F. S. Netto & L. F. Bittencourt (2004), Uma introdução à computação quântica, Trabalho de conclusão de curso, Universidade Federal do Paraná (UFPR).

Xi, M., J. Sun & W. Xu (2007), 'Quantum-behaved particle swarm optimization with elitist mean best position', *Complex Systems and Applications-Modeling, Control and Simulations* pp. 1643–1647.

Xi, M., J. Sun & W. Xu (2008), 'An improved quantum-behaved particle swarm optimization algorithm with weighted mean best position', *Applied Mathematics and Computation* **205(2)**, 751–759.

Xu, S. & Y. Rahmat-Samii (2007), 'Boundary conditions in particle swarm optimization revisited', *IEEE Transactions on Antennas and Propagation* **55**(3), 760–765.

Yasojima, C. T. K., W. A. L. Lobato & O. N. Teixeira (2009), Proposta de uma nova metaheurística híbrida com lógica nebulosa baseada em algoritmos genéticos com interação social, Trabalho de conclusão de curso, Centro Universitário do Pará (CESUPA).