Análise do Algoritmo de Otimização por Enxame de Partículas em Clãs em Problemas Dinâmicos com Domínio Contínuo

Rafael Henrique Vincence

Joinville

2017

Rafael Henrique Vincence

Análise do Algoritmo de Otimização por Enxame de Partículas em Clãs em Problemas Dinâmicos com Domínio Contínuo

Relatório de Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Graduação em Ciência da Computação, da Universidade do Estado de Santa Catarina (UDESC), como requisito parcial da disciplina de Trabalho de Conclusão de Curso.

Orientador: Profº Rafael Stubs Parpinelli

Joinville

2017

Rafael Henrique Vincence

Análise do Algoritmo de Otimização por Enxame de Partículas em Clãs em Problemas Dinâmicos com Domínio Contínuo

Relatório de Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Ciência da Computação da UDESC, como requisito parcial para a obtenção do grau de BACHAREL em Ciência da Computação.

Aprovado em 15 de Novembro de 2017

BANCA EXAMINADORA

Prof^{o} Rafael Stubs Parpinelli

Prof^{o} Claudio Cesar de Sá

Prof^{o} Chidambaram Chidambaram

Dedicatória

Agradeço, em especial, aos meus pais, Volney e Maria Janete, que sempre se esforçaram para que eu pudesse ter uma boa educação, sempre me apoiaram ao seguir o meu próprio caminho e assim chegar até aqui. Agradeço pelo apoio e também pela cobrança que recebi durante toda minha vida.

Agradeço ao meu orientador, Rafael Parpinelli, que desde o começo acreditou em mim e me deu todo o suporte necessário, mesmo nos muitos momentos difíceis que tivemos ao logo desse trabalho.

Agradeço aos amigos e colegas pelo apoio e a todos que de alguma forma contribuíram para este trabalho fosse realizado, em especial aqueles que me ajudaram com momentos de descontração e incentivo para nunca desistir.

Resumo

A maioria dos desafios encontrados na vida real são problemas complexos com uma grande gama de variáveis e por vezes dinâmico. Sendo assim, cada vez mais tem-se a necessidade da criação de meta heurísticas para solucionar problemas com estas características. A Natureza, por sua vez, resolve instintivamente esses tipos de problemas e por esse motivo algoritmos bioinspirados têm sido amplamente utilizados na resolução de problemas dinâmicos com domínio contínuo. Os aspectos e as interações de uma colônia de animais são pontos relevantes na otimização desses problemas, como o comportamento individual e comunitário, que contribuem em problemas de larga escala e não estacionários. Neste trabalho o foco é a análise e aplicação do algoritmo de otimização por exame de partículas (*Particle Swarm Optimization*) em *benchmarks* dinâmicos de domínio contínuo.

Palavras-chave: *Inteligência de enxames*, *Algoritmos bioinspirados*, *Problemas dinâmicos com domínio contínuo*, *Otimização por colônia de Partículas*.

Abstract

A maioria dos desafios encontrados na vida real são problemas complexos com uma grande gama de variáveis e por vezes dinâmico. Sendo assim, cada vez mais tem-se a necessidade da criação de meta heurísticas para solucionar problemas com estas características. A Natureza, por sua vez, resolve instintivamente esses tipos de problemas e por esse motivo algoritmos bioinspirados têm sido amplamente utilizados na resolução de problemas dinâmicos com domínio contínuo. Os aspectos e as interações de uma colônia de animais são pontos relevantes na otimização desses problemas, como o comportamento individual e comunitário, que contribuem em problemas de larga escala e não estacionários. Neste trabalho o foco é a análise e aplicação do algoritmo de otimização por exame de partículas (*Particle Swarm Optimization*) em *benchmarks* dinâmicos de domínio contínuo.

Key-Words: *Swarm Inteligence*, *Bioinspired Algorithms*, *dynamic problems with continuous domain*, *Particle Swarm Optimization*.

Conteúdo

Lista de Figuras

Lista de Quadros

Lista de Abreviaturas

BA	*Bat Algorithm*
BFO	*Bacterial Foraging Algorithm for Optimal*
CDE	*Crowding-based Diferencial Evolution*
CN	Computação Natural
DE	*Diferential Evolution*
EA	*Evolutionary Algorithms*
FA	*Firefly Algorithm*
FSS	*Fishing School Shearch*
MP	*Moving Peaks*
OP	*Ocillating Peaks*
OFFE	*Offline Error*
PSO	*Particle Swarm Optimization*
CPSO	*Clan Particle Swarm Optimization*
DCPSO	*Dynamic Clan Particle Swarm Optimization*
ClPSO	*Clustering Particle Swarm Optimizer*
VPSO	*Volitive Particle Swarm Optimizer*
DSPSO	*Dynamic Species-Based Particle Swarm Optimizer*
SDE	*Species-based Diferencial Evolution*
ShDE	*Sharing-based Diferencial Evolution*
SI	*Swarm Inteligence*

1 Introdução

Existem vários tipos de problemas que possuem uma grande dificuldade na obtenção de resultados satisfatórios em um tempo factível. Esses problemas tendem a ter uma grande dimensionalidade e serem não estáticos (OLIVEIRA; SILVA; ALOISE, 2004). Na Natureza pode-se observar uma constante mudança no ambiente, de forma que seus integrantes tenham que se adaptar a essas mudanças para poder sobreviver. Sendo assim, na Natureza existe uma grande fonte de inspiração para o desenvolvimento de novas tecnologias para solucionar problemas dinâmicos com domínio contínuo, particularmente na Ciência da Computação, dentro da área de Computação Natural, que é responsável pela criação de diversos algoritmos (CASTRO, 2007). A partir da observação e compreensão do comportamento de animais ou colônia de animais (fenômenos naturais) foram desenvolvidas várias tecnologias com diferentes aplicações (ROZENBERG; BCK; KOK, 2011), pois a natureza é capaz de trabalhar com problemas de alta complexidade de forma instintiva (ANDRÉ; STUBS PARPINELLI, 2015).

A característica que vem chamando atenção para a área de otimização na computação é justamente o fato de os problemas estarem em constante alteração e uma solução ótima, que foi encontrada em um instante de tempo anterior, pode não ser mais suficientemente boa para o mesmo problema no futuro. Isso é uma questão importante, pois quando um algoritmo de busca exaustiva é aplicado a um problema complexo, o tempo para encontrar uma solução é elevado, então não é recomendado a aplicação em problemas dinâmicos em que as soluções encontradas podem perder a validade com o decorrer do tempo (MORRISON, 2003). Por esse motivo, para otimizar esses problemas, são utilizados os algoritmos bioinspirados, que por sua vez possuem vários operadores, características relevantes que podem ser aplicados e estudados separadamente e assim estipular a relevância de cada um.

Na literatura são encontrados diversos exemplos de algoritmos com inspiração natural podendo se basear no comportamento de uma colônia de animais por busca de alimento, na interação que eles podem realizar entre si em sua colônia, ou até na própria evolução das espécies. Uma das primeiras meta heurísticas inspiradas na natureza, em especial na Biologia, são os Algoritmos Genéticos (*Genetic Algorithms*) (HOLLAND, 1975). Esta abordagem se baseia na teoria da evolução proposta por Darwin em que os indivíduos mais bem adaptados tem maiores chances de sobreviver e passar seu material genético adiante. Essa classe de algoritmos

é chamada de computação evolucionária e é composta por algoritmos evolucionários (*Evolutionary Algorithms* - EA). Outro algoritmo que entra nessa classe é o de Evolução Diferencial (*Diferential Evolution* - DE), que também possui uma rotina de seleção, cruzamento e mutação.

Dentre outros algoritmos bioinspirados existe uma classe que se baseia no comportamento simples de cada indivíduo separadamente e que em conjunto pode-se gerar um comportamento mais complexo, ou seja, um comportamento emergente. Essa classe é chamada de algoritmos de Inteligência de Enxame (*Swarm Inteligence* - SI) (PARPINELLI; LOPES, 2011). Entre eles pode-se citar algoritmos que têm sua aplicação em problemas contínuos, como o algoritmo baseado no comportamento de uma colônia de bactérias na busca por alimentos, o *Biomimicry of Bacterial Foraging Algorithm* (BFA) (PASSINO, 2002); e o algoritmo baseado no comportamento das colônias de vaga-lumes e sua bioluminescência que guia os outros vaga-lumes no espaço de busca, o *Firefly Algorthm* (YANG, 2008). Existem algoritmos inspirados no comportamento coordenado do movimento de cardumes de peixes e de revoada de pássaros, o Algoritmo de Otimização por Enxame de Partículas (*Partical Swarm Optimization*, PSO) (EBERHART; KENNEDY et al., 1995). Há também algoritmos que misturam operadores de colônia de animais diferentes, como uma versão do PSO que usa um operador de expansão e contração de um cardume de peixes (um operador de movimentação volátil) aplicado no PSO para melhorar o desempenho em problemas dinâmicos, sendo criado uma nova versão, o *Volitive* PSO (CAVALCANTI-JÚNIOR, 2011), que obteve bons resultados aplicado a essa classe de problemas.

O algoritmo que é o foco deste trabalho é o Algoritmo de Otimização por Enxame de Partículas por Clãs (*Clan Partical Swarm Optimization*, CPSO) (CARVALHO; JOSÉ ALBANEZ BASTOS-FILHO, 2009), uma versão baseada no PSO que utiliza sub-populações chamadas de clãs e seleciona líderes nesses clãs para uma troca de informações. As suas principais características são sua simplicidade (utilizando as características do PSO padrão em cada um dos clãs) e conseguir manter o rastreamento de vários pontos no espaço de busca ao mesmo tempo.

A literatura apresenta uma vasta quantidade de trabalhos que estudam a aplicação desses algoritmos bioinspirados em problemas dinâmicos com domínio contínuo. Entre eles pode-se citar a aplicação do Algoritmo de Evolução diferencial com as versões baseadas em aglomeração (*Crowding-based DE* CDE), baseado em compartilhamento (*Sharing-based DE* ShDE) (THOMSEN, 2004), e a versão baseada em espécies (*Species-based DE* SDE) (LI, 2005). Essas versões do DE foram estudadas e comparadas para que o *Crowding-based local Differential Evolution with Speciation-based Memory* (ClDES) fosse desenvolvido utilizando seus pontos positivos. Na área de Inteligência de enxames são encontrados várias versões do PSO,

como por exemplo: *Dynamic Species-Based Particle Swarm Optimizer* DSPSO (PARROTT; LI, 2006) e o *Clustering Particle Swarm Optimizer* CPSO (YANG; LI, 2010). Existe também os algoritmos de (*Dynamic Bacterial Foraging Algorithm* - DBFA) proposto por (PASSINO, 2002) e o *Multiswarm Based Firefly Algorithm* proposto por (FARAHANI; NASIRI; MEYBODI, 2011).

Como não se pode ter uma solução ótima a todo momento em problemas dinâmicos, o tempo de execução é considerado como uma unidade discreta. Geralmente um dos fatores limitantes em uma aplicação acaba sendo o tempo de execução sendo necessário manter um equilíbrio entre a qualidade da solução e o tempo de execução do algoritmo (LI; YANG; KANG, 2006). Apesar dos algoritmos bioinspirados serem capazes de encontrar boas soluções para problemas reais, eles tendem a perder a eficiência quando aplicados a problemas de larga escala. Esta característica indesejável é conhecida por "maldição da dimensionalidade" (*curse of dimensionality*) (BELLMAN; DREYFUS, 2015). Isso ocorre devido ao crescimento exponencial do espaço de busca de acordo com as dimensões do problema.

O grande número de dimensões aumenta a dificuldade dos algoritmos em manter soluções aceitáveis. A qualidade da otimização de um algoritmo depende do equilíbrio dos componentes de diversificação e intensificação (BOUSSAÏD; LEPAGNOT; SIARRY, 2013), pois a diversificação é responsável pela exploração do espaço de busca como um todo e a intensificação é responsável pela acurácia da resposta. A natureza evoluiu para manter o equilíbrio de diversidade e na otimização é possível usar alguns componentes de controle para sua preservação. A literatura aponta diversas estratégias para este controle de diversificação, sendo algumas: *fitness sharing, clearing, crowding, deterministic crowding, probabilistic crowding* e *generation gap* (ANDRÉ; STUBS PARPINELLI, 2015).

Na literatura, os algoritmos bioinspirados são aplicados a diversos tipos de problemas dinâmicos, sendo eles contínuos ou discretos. Para poder avaliar os algoritmos, utiliza-se diversas funções *benchmarks* (MOSER, 2007), como por exemplo a função de Movimentação de Picos (*Moving Peaks* - MP). A qualidade da solução sendo otimizada pode ser mensurada pela sua aptidão, ou seja, quão interessante é o valor encontrado e quão bem o algoritmo se adapta a uma mudança do ambiente. Assim, uma solução com uma boa aptidão (*fitness*) durante o processo de evolução é considerada válida e é chamada de solução factível. Durante a otimização dos *benchmarks* podem aparecer diferentes tipos de dinamismo, como por exemplo: na função objetivo; no domínio das variáveis; no número de variáveis; nas restrições; ou outros parâmetros.

O motivo da escolha do CPSO como foco do trabalho se deve ao amplo estudo que existe do PSO em problemas dinâmicos e o fato dessa versão, em clãs, funcionar do mesmo modo que o PSO canônico em cada um das subpopupações, ou seja, fácil de se aplicar operadores que já foram aplicados em outras versões do PSO. Na literatura, quando um algoritmo é aplicado em problemas dinâmicos complexos é feita uma adaptação deste algoritmo e assim é gerado uma nova versão com novas características. O CPSO por padrão já possui algumas características, como manutenção da diversidade por clãs.

Na realização desse trabalho foram estipulados objetivos para determinar os passos a serem seguidos, na próxima seção serão apresentados esses objetivos para estruturar os capítulos do trabalho.

1.1 Objetivo

Este trabalho tem como objetivo analisar e comparar da eficiência do CPSO e seus operadores em problemas dinâmicos com domínio contínuo. A proposta central é analisar cada um dos operados evolutivos do algoritmo e determinar a relevância de cada um para a otimização dessa classe de problemas. Para isso será feito uma análise comparativa dos algoritmos que tem relevância na otimização desses problemas, indicando os pontos positivos e negativos de cada um. Para atingir o objetivo principal alguns objetivos específicos foram traçados:

- Revisão bibliográfica dos conceitos da computação natural e meta heurísticas bioinspiradas;
- Análise aprofundada do CPSO e seus operadores;
- Revisão bibliográfica dos operadores evolutivos existentes;
- Estudo dos algoritmos evolutivos de inteligência de enxame para verificar a relevância dos operadores existentes e realizar uma análise comparativa
- Levantamento e descrição de funções dinâmicas para serem aplicadas nos experimentos;
- Implementar o CPSO aplicando operadores evolutivos diversos;
- Realizar experimentos computacionais com o algoritmo e os problemas selecionados;

- Coleta e análise dos resultados dos experimentos.
- Exploração dos parâmetros do algoritmo para avaliar o desempenho em vários casos.
- Apresentar os resultados comparando com outras versões do PSO e outros algoritmos.

1.2 Estrutura do Trabalho

O trabalho está organizado em 7 Capítulos, incluindo a introdução como primeiro capítulo.

No segundo capítulo é feita uma revisão da definição de intensificação e diversificação, em seguida é feito uma revisão do estado da arte dos algoritmos bioinspirados e as suas características principais, sendo separados em dois grupos: os algoritmos evolutivos e os de inteligência de enxames. Após isso é feito uma revisão da diversidade populacional e métricas encontradas na literatura. Por fim é apresentado uma definição dos problemas que serão estudados neste trabalho, esquematizando suas características principais, suas dificuldades e métodos de avaliação para o algoritmo que será utilizado para realizar a otimização.

No terceito capítulo é apresentada uma revisão dos trabalhos encontrados nessa área de pesquisa, onde pode ser analisado outras aplicações em problemas dinâmicos e com isso definir quais os pontos mais relevantes e fornecer parâmetros para a comparação dos resultados.

No quarto capítulo é descrita o modelo proposto e uma detalhação das implementações feitas para obtenção dos resultados.

No quinto capítulo terá o protocolo de experimentação utilizado junto com um sumarização dos experimentos feitos para testar os limites do algoritmo.

No sexto capítulo será apresentado os resultados obtidos comparando com os algoritmos encontrados na literatura e os resultados de experimentos feitos alterando os parâmetros do sistema.

Por fim, no sétimo capítulo encerra-se o trabalho com uma breve revisão das principais considerações, apresentando uma discussão dos resultados obtidos com a pesquisa e aponta trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são abordados conceitos importantes para o entendimento das técnicas que são utilizadas neste trabalho. A Seção 2.1 apresenta uma introdução sobre diversificação e intensificação, na Seção 2.2 é introduzido os algoritmos evolutivos, apresentando seus pontos fortes e suas influências para a otimização de problemas complexos. A seguir, na Seção 2.3 é introduzido os algoritmos de inteligência de enxame e seus pontos positivos na otimização de problemas dinâmicos. Por fim, na Seção 2.4 é apresentado a importância da diversidade populacional, mostrando métodos de manutenção e métricas de avaliação.

2.1 Intensificação e Diversificação

A qualidade da otimização de um algoritmo depende do equilíbrio dos componentes de diversificação e intensificação (ČREPINŠEK; LIU; MERNIK, 2013). O equilíbrio adequado permite explorar com mais eficiência o espaço de busca e, consequentemente, obter resultados melhores. A diversificação explora o máximo possível do espaço de busca para identificar diferentes picos de soluções de alta qualidade. Na intensificação ocorre a exploração local, em que se faz a busca mais intensiva em uma área específica. Um algoritmo com rotinas de diversificação em excesso se aproxima de um processo de busca aleatória e um com rotinas de intensificação em excesso leva a uma maior chance de estagnar a busca em um ponto subótimo.

Esses dois componentes têm ligação direta com a diversidade populacional, de modo que um sistema com muita intensificação gera pouca diversidade. Na Natureza pode-se observar que espécies de animais com grande diversidade tem maiores chances de sobreviver, e para problemas dinâmicos a diversidade é um ponto ainda mais crucial para manter um certo nível de aptidão dos indivíduos durante o processo de otimização.

2.2 Algoritmos Evolutivos

A natureza é uma fonte de inspiração para o desenvolvimento de vários algoritmos, como por exemplo os algoritmos evolutivos. A Computação Evolutiva se inspira no processo

da seleção natural e evolução natural. Em termos gerais, um algoritmo evolucionário possui alguns componentes básicos para resolução de problemas (PARPINELLI; LOPES, 2011):

1. Representação das possíveis soluções do problema;
2. Um modo de criar uma população inicial (aleatório ou determinístico);
3. Uma função que avalia a qualidade das soluções, ou seja, a aptidão do indivíduo (*Fitness*);
4. Um mecanismo de seleção para cruzamento;
5. Operadores evolutivos para criação de novas gerações (como mutação e *crossover*);
6. Parâmetros para controle do comportamento do algoritmo, como taxa de cruzamento, taxa de mutação, etc.

Nas subseções são apresentados os algoritmos do estado da arte no contexto deste trabalho.

2.2.1 Algoritmo Genético

O Algoritmo Genético (*Genetic Algorithm* - GA) foi um dos primeiros algoritmos bioinspirados a ser proposto durante as décadas de 60 e 70. Ele foi desenvolvido por Holland e seus colaboradores que tem como base a teoria da evolução de Darwin (BOOKER; GOLDBERG; HOLLAND, 1989) e é um dos algoritmos mais utilizados na Computação Evolutiva. A seleção natural de Darwin diz que o melhor indivíduo em uma determinada população tem maiores chances de sobreviver e assim passar sua carga genética adiante, tornado a espécie mais apta às condições do ambiente. O GA utiliza essa seleção do indivíduo mais adaptado para direcionar a busca para regiões promissoras no espaço de busca.

A otimização do GA começa com a inicialização da população inicial, sendo de forma aleatória ou determinística, em que cada indivíduo possui um cromossomo, que por sua vez representa uma possível solução para o problema. A partir dessa população inicia-se o primeiro ciclo evolutivo, em que cada indivíduo é avaliado, gerando assim um valor de aptidão (*fitness*) para ser usado na seleção da população. A seleção determina quais indivíduos irão cruzar gerando uma população intermediária, de modo que os mais adaptados possuem uma maior chance de serem selecionados. Os operadores evolutivos de cruzamento e mutação são

aplicados na população intermediária, em que o cruzamento tem o papel de intensificar a busca e a mutação o de diversificar a população. A partir deste ciclo o algoritmo se repete até que o critério de parada seja satisfeito.

2.2.2 Evolução Diferencial

A evolução diferencial (*Diferencial Evolution* - DE) é uma meta-heurística evolucionária para otimização de funções contínuas, proposta por Storn e Price em 1995 (PRICE; STORN; LAMPINEN, 2006). Seu nome se dá pela sua rotina de mutação que acontece por uma operação de subtração (diferenciação). O DE é amplamente estudado e as características que o tornam interessante são: Sua simplicidade, pois é um algoritmo com poucos operadores evolutivos, simples de ser implementado; Seu bom desempenho, tendo uma convergência rápida. O DE se destaca entre vários outros algoritmos evolutivos; e pelo fato de possuir poucos parâmetros, tornando assim sua aplicação mais fácil e intuitiva.

O DE possui, basicamente, uma etapa de inicialização e três etapas em seu ciclo de evolução: mutação, cruzamento e seleção (nessa ordem). O processo evolutivo é iniciado após a inicialização do algoritmo e é finalizado quando um determinado critério de parada for atingido. Primeiramente, o usuário define o tamanho da população (λ), taxa de chance de *crossover* (C_r) e fator de escala (F). Existem três tipos de vetores no DE que são: Vetor alvo, que é o vetor pai da geração atual; O vetor doador, que é gerado a partir da mutação; e o vetor *trial*, que é a combinação do vetor alvo com o vetor doador gerado pelo cruzamento.

Para criar o vetor doador, obtém-se uma amostra de três indivíduos distintos na população. Assim, é criado uma mutação a partir da diferença de dois vetores, multiplicada pelo fator de escala F e somada ao terceiro vetor. A etapa de mutação, responsável pela busca global do algoritmo (diversificação) é o principal fator caracterizante do DE. O C_r controla o cruzamento que é responsável pela intensificação da busca e acontece a partir da recombinação dos vetores alvo e doador. Ao final, a seleção acontece verificando uma melhora no *fitness* do vetor de cruzamento em relação ao alvo, caso não aconteça uma melhora a nova solução é descartada.

2.3 Algoritmos de Inteligência de Enxame

Sistemas baseados em enxames são inspirados pelo comportamento de alguns seres vivos sociais, como formigas, cupins, aves e peixes. A auto-organização e controle descentralizado são características notáveis de sistemas baseados em enxames que, tal como na natureza, levam a um comportamento emergente. Este comportamento é uma propriedade que emerge através de interações locais entre os componentes do sistema e não é possível de ser conseguido por qualquer um dos componentes do sistema atuando isoladamente (GARNIER; GAUTRAIS; THERAULAZ, 2007).

2.3.1 Otimização por Colônia de Bactérias

O algoritmo de Otimização por Colônia de Bactérias (*Bacterial Foraging Algorithm* - BFA) foi proposto por Passino (PASSINO, 2002), inspirado no comportamento social de busca por alimento das bactérias Escherichia Coli. Durante o processo de busca a bactéria se move em pequenos passos enquanto procura alimento. Seu movimento é realizado através de um conjunto de filamentos conhecido como flagelos, que ajudam a bactéria se mover em períodos alternados de natação (*swim*) e tombos (*tumble*). A alternância entre estes dois períodos chama-se quimiotaxia. Cada bactéria representa uma solução para o problema. O ambiente fornece o substrato para as bactérias interagirem e é representado pelo espaço de busca sendo otimizado. Quanto melhor for a região do espaço de busca, melhor será o resultado da função objetivo, e consequentemente, melhor será o substrato para as bactérias.

O BFA é composto por três rotinas principais: quimiotaxia, reprodução e eliminação-dispersão. Na quimiotaxia, uma bactéria com direção aleatória representa um tombo e uma bactéria com a mesma direção do passo anterior indicando uma execução. Na reprodução, a saúde de cada bactéria representa seu valor de *fitness*. Todas as bactérias são classificadas de acordo com seu estado de saúde e apenas a primeira metade da população sobrevive. As bactérias sobreviventes são divididas em dois filhos idênticos, de modo a formar uma nova população. O processo de eliminação-dispersão é responsável por aumentar a diversidade da população. A dispersão ocorre depois de um certo número de etapas de reprodução, quando algumas bactérias são escolhidas de acordo com uma probabilidade predefinida. Tais bactérias são mortas e os novos são gerados aleatoriamente em outra posição dentro do espaço de busca. A intensificação da busca é realizada por ambos os passos de quimiotaxia e de reprodução.

2.3.2 Otimização por Colônia de Vaga-Lumes

O algoritmo de Otimização por Colônia de Vaga-Lumes (*Firefly Algorithm* - FA) foi proposto por Yang (YANG, 2008). O FA trabalha com 3 regras principais para a otimização, sendo elas:

- Um vaga-lume será atraído pelo outro não importando o sexo deles;
- A atratividade entre dois vaga-lumes aumenta em relação à intensidade do brilho e decresce em relação ao aumento da distância entre eles;
- A proximidade do vaga-lume em relação a uma solução do espaço de busca influência na intensidade do brilho do mesmo.

Cada agente brilha proporcionalmente à qualidade da sua solução que, juntamente com a sua atratividade (β), ditam o quão forte ela atrai outros membros do enxame. Dois outros parâmetros definidos pelo usuário são o valor máximo de atração (β_i) e do coeficiente de absorção (Υ) que determina a variação de atratividade com o aumento da distância da comunicação dos vaga-lumes. A variável de intensidade da luz β_i de cada vaga-lume, mantém o balanço entre a intensificação e diversificação da busca.

2.3.3 Otimização por Colônia de Morcegos

O algoritmo de Otimização por Colônia de Morcegos foi apresentado pela primeira vez por Yang (YANG; HOSSEIN GANDOMI, 2012). O algoritmo é inspirado no processo de ecolocalização dos morcegos utilizado durante o seu voo para detectar presas e evitar obstáculos. Além disso é utilizada uma rotina de voo aleatório para movimentação no espaço de busca, atualizando suas posições e suas velocidades. Outros parâmetros são: o fator de decaimento de sonoridade (α) que atua em um papel semelhante a temperatura do *simulated annealing*, e o fator de aumento de pulso (γ) que regula a frequência de pulso. A atualização da taxa de pulso (R_i) e sonoridade (A_i) equilibra o comportamento intensificação e diversificação de cada morcego, respectivamente. De acordo com a diminuição da sonoridade a taxa de pulso vai aumentando para melhorar a acurácia do ataque.

2.3.4 Otimização por Enxame de Partículas

A otimização por enxame de partículas (*Particle Swarm Optimization* - PSO) foi proposta por Kennedy (EBERHART; KENNEDY et al., 1995). O PSO tem como inspiração o comportamento coordenado dos movimentos dos pássaros e cardumes de peixes. Cada partícula é uma solução potencial para o problema, representada por sua velocidade e posição, localização no espaço de busca e uma memória que armazena a sua melhor posição visitada. O movimento de cada partícula usa uma combinação do seu próprio vetor de velocidade atual sendo atraída para sua melhor posição e para a melhor posição encontrada por toda a população. Cada iteração do algoritmo ocorre uma atualização na velocidade de cada partícula utilizando a seguinte equação:

$$\begin{aligned} \vec{V_{id}}(t+1) &= \vec{V_{id}}(t) + cognitive + social \\ cognitive &= c_1\varepsilon_1(\vec{P_{id}}(t) - \vec{X_{id}}(t)) \\ social &= c_2\varepsilon_2(\vec{P_{gd}} - \vec{X_{id}}(t)) \end{aligned} \tag{2.1}$$

- Onde $\vec{V_{id}}(t)$ é a velocidade da partícula i com dimensão d na iteração t. Que influência na direção tomada pela partícula para percorrer o espaço de busca e evitar mudanças drásticas na direção atual da partícula.

- O $cognitive$ é chamado de componente cognitivo que utiliza a melhor posição encontrada pela partícula em questão $\vec{P_{id}}(t)$. Ele representa o quanto a partícula irá se movimentar, começando na sua posição $\vec{X_{id}}(t)$, em direção a sua melhor posição $\vec{P_{id}}(t)$.

- O $social$ é o componete responsável pela comunicação das partículas entre si, de forma que o melhor resultado encontrado na população $\vec{P_{gd}}$ é usado para influencias cada indivíduo da população. Ele representa o quanto a partícula irá se movimentar, começando na sua posição $\vec{X_{id}}(t)$, em direção a sua melhor posição da população $\vec{P_{gd}}(t)$.

- c_1 e c_2 são constantes de aceleração positivas. São utilizadas para quantificar a contribuição do movimento cognitivo e social, respectivamente, na população.

- ε_1 e ε_2 são números aleatórios entre 0 e 1 gerados por uma distribuição normal.

O equilíbrio entre a intensificação e a diversificação é alcançado no PSO através do comportamento social e o cognitivo, respectivamente. Utilizando o valor obtido da velocidade é feito a atualização da posição $t + 1$ usando a equação:

$$\vec{X_{id}}(t+1) = \vec{X_{id}}(t) + \vec{V_{id}}(t+1) \tag{2.2}$$

O PSO pode ter definido um valor V_{max} de velocidade máxima para evitar um estado de explosão da velocidade. Esse estado pode ocorrer por causa do valor de c_1 e c_2 que pode induzir a um rápido crescimento da velocidade. Uma abordagem do PSO utiliza outro coeficiente na equação de atualização da valocidade 2.1. Esse outro coeficiente é chamado de fator de inércia **??** representado pelo ω na equação:

$$\vec{V_{id}}(t+1) = \omega\vec{V_{id}}(t) + cognitive + social \tag{2.3}$$

O fator de inércia ω controla a habilidade de exploração e intensificação na busca balanceando o impulso da partícula aplicando um peso no valor atual da velocidade $\vec{V_{id}}(t)$. Esse balanceamento funciona diminuindo o valor da velocidade em cada etapa do algoritmo, fazendo com que o PSO melhore seu nível de exploração. Outro modelo de PSO proposto utiliza um fator de constrição na equação de atualização da velocidade:

$$\vec{V_{id}}(t+1) = \chi[\vec{V_{id}}(t) + cognitive + social] \tag{2.4}$$

$$\begin{aligned} \chi &= \frac{2k}{2 - \varphi\sqrt{\varphi^2 - 4\varphi}} \\ \varphi &= c_1\varepsilon_1 + c_2\varepsilon_2 \end{aligned} \tag{2.5}$$

Tendo como restrições $\varphi \geq 4$ e $k \in [0, 1]$ o fator de constrição pode garantir uma convergência da busca. Utilizando esses valores o fator χ assume um valor entre 0 e 1 que implica em uma redução na velocidade em cada etapa do algoritmo. O parâmetro k na equação 2.5 controla a exploração e a intensificação do enxame, sendo que, valores próximos a 1 são usados para melhorar a exploração tendo uma convergência mais lenta e para se obter uma convergência mais acelerada, é necessário diminuir o valor de k.

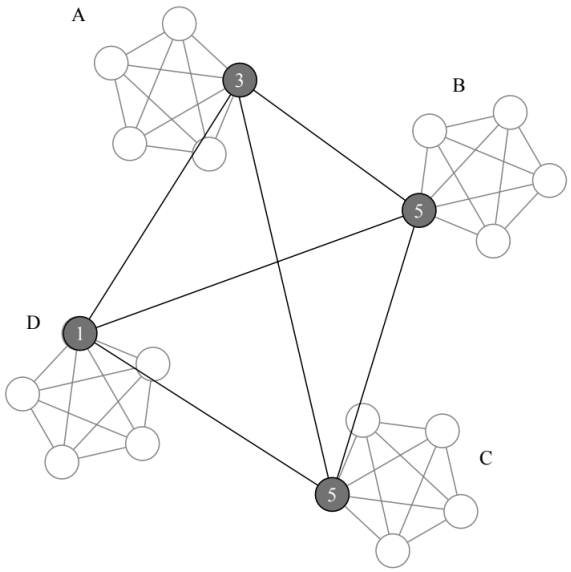
2.3.5 Otimização por Enxame de Partículas em Clãs

O algoritmo de otimização por enxame de partículas em clãs (*Clan Particle Swarm Optimization*, CPSO) (CARVALHO; JOSÉ ALBANEZ BASTOS-FILHO, 2009) é um algoritmo derivado do PSO que trabalha com subpopulações chamadas de clãs. A inspiração para esse algoritmo vem do comportamento animal de bandos, onde em um espaço de busca com o objetivo de obtenção de alimentos várias populações podem buscar por pontos ótimos em separado.

A separação do clã acontece simplesmente dividindo a população em um número de clãs de modo que nem um dos clãs tenha mais indivíduos que o outro. Normalmente essa divisão ocorre sequencialmente, porém existem técnicas de *clustering* que auxiliam nessa separação, para que em cada clã se juntem os indivíduos mais parecidos/próximos, assim melhorando a diversidade entre os clãs.

O processo de evolução acontece da mesma forma que o PSO padrão, com o cálculo da velocidade e com a atualização das posições, porém isso é feito tomando cada clã como um população, de forma que o melhor global seja o melhor do clã. Após cada etapa de atualização das velocidades e de posição de cada indivíduo, é feito a avaliação do *fitness* e definido um líder, de forma que o indivíduo de cada clã com o maior *fitness* é o líder.

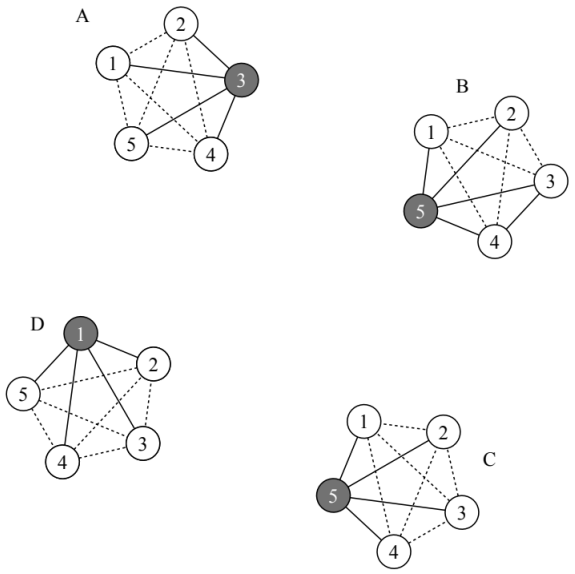
Figura 2.1: representação de uma população com 4 clãs e seus líderes



Quando todos os líderes forem definidos é criado uma nova população somente com eles e acontece mais uma etapa de evolução, atualizando a velocidade e a posição deles uma segunda vez nessa iteração, como é mostrado na Figura 2.1. Essa etapa de evolução dos líderes é

chamado de conferência do líderes (*leader conference*) e tem como objetivo trocar informações entre eles. Por causa dessa última etapa esse algoritmo acaba tendo mais avaliações de *fitness* por ciclo do que o número de indivíduos da população, sendo esse valor mais o número de clãs.

Figura 2.2: representação dos líderes disseminando a informação



No começo de cada nova etapa, os líderes de cada clã serão classificados como ótimo global de seus respectivos clãs e poderão disseminar a informação obtida através da conferência dos líderes, como é mostrado na Figura 2.2. Assim o processo se repete e após cada evolução dos clãs é definido um novo líder.

Um ponto negativo desse processo de separação em clãs, é que existe o risco dessas subpopulações se agruparem, assim perdendo o monitoramento de vários pontos ótimos ao mesmo tempo, e na sua versão padrão não existe uma rotina para evitar isso ou até mesmo dispersar a população, uma vez que foi identificado o agrupamento dos clãs. As etapas do processo de evolução podem ser identificadas no algoritmo 1.

```
InicializarPopulação();
while numeroIteração <= totalIterações do
    for clãs do
        for partículas do
            AtualizarVelocidade();
            AtualizarPosição();
            CalcularFitness();
            AtualizarInformações();
        end
        DelegarLider();
    end
    RealizarConferênciaLideres();
    numeroIteração ++;
end
```

Algorithm 1: Pseudo-código do Algoritmo de Otimização de Partículas por Clãs

2.4 Diversidade Populacional

Uma população com diversidade pode ser capaz de lidar com funções multimodais e explorar simultaneamente diferentes picos na superfície da função objetivo, como também melhorar a exploração global e ajudar a encontrar vários ótimos globais e locais.

Em problemas dinâmicos uma boa diversidade populacional ajuda o sistema a manter um nível satisfatório de *fitness* durante todo o processo de otimização. Ajuda também o sistema a se adaptar as mudanças no ambiente, de forma que, ao perceber essa mudança os indivíduos mais afastados dos pontos ótimos podem encontrar diferentes soluções que podem se tornar novos pontos ótimos.

Além disso, muitos dos problemas dinâmicos possuem mais de um ponto ótimo e algumas estratégias de manutenção da diversidade populacional, mostradas na Seção 2.4.1 ajudam a encontrar vários pontos ao mesmo tempo.

2.4.1 Manutenção

As principais estratégias encontradas na literatura são o *fitness sharing, clearing, crowding, deterministic crowding* e *probabilistic crowding*.

Compartilhamento (*Sharing*): Foi introduzido por Goldberg e Richardson (GOLDBERG; RICHARDSON, 1987), com a intenção de localizar múltiplos picos simultaneamente. A ideia básica é punir indivíduos que ocupam as mesmas regiões no espaço de busca por escala a aptidão de cada indivíduo de acordo com o número de indivíduos na vizinhança. Assim, penalizando indivíduos que se aglomeram em conjunto, o regime de partilha impede a convergência para um único pico. O *fitness sharing* de um indivíduo i é definido pela Equação 2.6. Onde n é o número de indivíduos da população, σ_{share} indica o limiar de dissimilaridade e d_j^i é a medida de distância entre o indivíduo i e j.

$$
\begin{aligned}
f_{share}^i &= \frac{f_i}{\sum_{j=0}^{n} sharing_j^i} \\
sharing_j^i &= \begin{cases} 1 - (d_j^i/\sigma_{share})^2 & \text{Se } d_j^i < \sigma_{share} \\ 0 & \text{Se não} \end{cases}
\end{aligned}
\tag{2.6}
$$

Compensação (*clearing*): é um método similar ao Compartilhamento, porém é baseado no conceito de recursos limitados no ambiente (PÉTROWSKI, 1996). Ao invés de compartilhar recursos entre todos os indivíduos de uma subpopulação, o método de compensação atribui o *fitness* apenas para os melhores indivíduos da subpopulação. Assim, o *clearing* preserva o *fitness* dos k melhores indivíduos do nicho e redefine o *fitness* dos outros que pertencem à mesma subpopulação. Como no método de *sharing*, indivíduos pertencem ao mesmo nicho se sua distância no espaço de busca for menor que um limiar de dissimilaridade σ_s (raio de *clearing*). O *clearing* pode ser combinado com estratégias de elitismo para preservar as melhores soluções dos nichos durante gerações.

Aglomeração (*Crowding*): O método de *Crowding* foi proposto por De Jong (DE JONG, 1975). Este método de nichos de espécies compara um descendente com alguns indivíduos escolhidos aleatoriamente da população atual. O indivíduo mais semelhante será substituído se o descendente oferecer melhor valor de *fitness*. O fator de aglomeração (parâmetro CF) é definido como uma pequena parcela da população, o qual é utilizado para controlar o tamanho da amostra. Nesse método existe a possibilidade de se sobrescrever um mesmo indivíduo várias vezes, esse fato se chama erro de substituição que é a principal desvantagem do

Crowding, além disso, o *Crowding* é propenso a aglomeração de indivíduos ao redor dos picos de soluções ótimas.

Aglomeração Determinística (*Deterministic Crowding*): Para aliviar o problema de erro de substituição, foi introduzido uma melhoria na aglomeração original, que não utiliza o parâmetro do fator de aglomeração. Este método foi chamado aglomeração determinista (MAHFOUD, 1995). Essa abordagem minimiza o erro de substituição significativamente e restaura a pressão seletiva. No entanto, este método também tem uma desvantagem em que aumenta a perda de nichos devido ao uso de seleção por torneio entre indivíduos semelhantes.

Aglomeração Probabilística (*Probabilistic Crowding*): Para superar o problema da falta ótimos locais foi criado o método de aglomeração probabilística (MENGSHOEL; GOLDBERG, 1999). A regra de substituição probabilística é usada para manter uma elevada diversidade na população que substitui indivíduos de menor *fitness* por indivíduos de *fitness* mais elevados de acordo com a sua proporção de *fitness*. O principal problema deste processo é que ele tem uma taxa de convergência muito lenta e baixa capacidade de pesquisa.

Lacuna de Geração (*Generation Gap*): Neste modelo apenas uma parte da população (*generation gap*) se reproduz e morre em cada geração. Deste modo o filho substitui o indivíduo mais similar retirado de uma subpopulação aleatória de tamanho CF (*crowding factor*) da população total.

2.4.2 Métricas

Para avaliar o estado da diversidade de uma população são utilizados dois tipos de medidas de diversidade: A medida de diversidade fenotípica (MDF) e genotípica (MDG). A MDF mede a diversidade de *fitness* das soluções, ou seja, a diversidade de diferentes qualidades de soluções. Já a MDG mensura a diversidade genética das soluções, avaliando o conjunto de variáveis (genes) que influenciam na função objetivo.

A MDF é apresentada na equação 2.7, em que variável VMD é um fator de normalização que corresponde a medida de diversidade de uma população virtual com os valores de *fitness* distribuídos de maneira uniforme com uma distância predeterminada. Onde f_{worst} representa o pior *fitness* e o f_{best} o melhor *fitness* e N o tamanho da população (CORRIVEAU, 2013). Quando o tamanho da população ou a faixa de *fitness* absoluto são modificados é necessário recalcular o VMD. É importante ressaltar que para calcular a MDF é preciso que os

valores de *fitness* estejam ordenados.

$$MDF = \frac{\sum_{i=0}^{n-1} \ln(1 + |f_i - f_{i+1}|)}{VMD} \tag{2.7}$$

A MDG proposta por Corriveau (CORRIVEAU, 2012). Esta medida foi desenvolvida com base na MDF, sendo indicada para variáveis contínuas. A medida é apresentada na Equação 2.8, onde D é a quantidade de dimensões da solução, N o tamanho da população e x a solução candidata. A variável $NMDF$ é o fator de normalização que corresponde ao valor máximo de diversidade conhecida até o momento. O valor 1 obtido da medida corresponde a diversidade genotípica máxima enquanto o valor 0 corresponde a convergência da população de soluções.

$$MDG = \frac{\sum_{i=0}^{n-1} \ln\left(1 + \min_{j \in [i_1, N]} \frac{1}{D} \sqrt{\sum_{k=1}^{D} (x_{i,k} - x_{j,k})^2}\right)}{NMDF} \tag{2.8}$$

2.5 Aplicação do AG em ambientes dinâmicos

Um trabalho que pode ser citado em relação a avaliação de algoritmos em ambientes dinâmicos é o trabalho de (RAND; RIOLO, 2005), em que é feita uma análise do comportamento do AG em um ambientes dinâmicos discreto. Mesmo sendo um trabalho que é aplicado em uma classe de problemas diferente da abordada por este trabalho, ele mostra que na maioria dos trabalhos que analisam um algoritmo somente avaliam o desempenho, ou seja, o quão perto do melhor resultado chegou. São analisados quatro fatores principais para determinar a eficiência do AG em ambientes dinâmicos, sendo eles:

1. Desempenho: Para entender o desempenho do algoritmo existem duas vertentes, sendo uma a avaliação do melhor indivíduo da população para cada iteração do AG, e a outra é avaliar a média da população em para cada uma das iterações. Para a aplicar AG no SL-HDF é usado a melhor solução antes da primeira alteração como média inicial.

2. Satisfatibilidade: É a medida da habilidade do sistema de manter um certo nível de *fitness* no decorrer da otimização e não deixar esse nível cair abaixo de um determinado limite. Esta medida não necessariamente representa o quão rápido (menos interações necessárias) o sistema chega em uma nova ótima solução, e sim se ele consegue manter um nível de *fitness* da população.

3. Robustez: É a medida de como o sistema reage a uma alteração, de forma que ao sofrer uma alteração o *fitness* não pode ter uma queda muito brusca. A medida de robustez usada neste trabalho foi a média do *fitness* no estado atual do ambiente sobre a média do *fitness* no estado anterior do sistema, para uma alteração perceptível.

4. Diversidade: É a medida que representa a variação do genoma da população, de forma que uma população que possui uma alta diversidade tem maiores chances de encontrar novas solução e assim se adaptar melhor a uma mudança do ambiente. Existem várias técnicas estudadas para manter a diversidade da população durante o processo evolutivo, e para medir a diferença genotípica é usado a distância de *Hamming*.

Na aplicação do AG pode-se notar que a performance do algoritmo no ambiente dinâmico é superior sua aplicação em ambientes estáticos quando há um grande número de iterações. Inicialmente o ambiente dinâmico perde para o estático mas a partir da metade do processo evolutivo o dinâmico gera um *fitness* maior no melhor indivíduo e na média da população.

A análise da satisfatibilidade mostra o nível do *fitness* durante o processo evolutivo, e com ele pode-se notar que mesmo em relação a uma aplicação em um ambiente estático, o nível de *fitness* pode ser mantido acima de um limiar.

Na análise de robustez pode-se notar que a cada mudança no ambiente existe uma queda na média do *fitness*, porém o sistema se recupera rapidamente, e a cada nova mudança e queda do *fitness* diminui.

A diversidade do sistema teve um comportamento inesperado, pois inicialmente achava-se que o sistema iria perder a diversidade até uma mudança ocorrer e depois a diversidade iria aumentar, porém aconteceu exatamente o contrário, tendo que após uma mudança a diversidade diminui e vai aumentando até identificar uma nova mudança.

2.6 Problemas Dinâmicos com Domínio Contínuo

Neste capítulo é apresentado uma breve introdução aos *benchmarks* que serão utilizados para análise nesse trabalho, junto a isso é feita uma introdução aos problemas dinâmicos, suas variações e dificuldades.

Durante o dia a dia, pode-se observar diversos problemas complexos em que as

condições do ambiente podem mudar com o tempo, sendo assim uma solução válida encontrada em um instante anterior acaba não sendo mais suficientemente boa para suprir as necessidades das novas condições (BRANKE, 2012). Para problemas com mudanças em seu ambiente é necessário um sistema de resolução que se adapte as alterações e esteja sempre apto a procurar por soluções melhores. Na computação existem várias áreas nas quais se dedicam a otimizar esses problemas, como por exemplo os EA e os SI. Para poder avaliar os algoritmos utilizados na otimização destes problemas são geradas funções *benchmarks* que apresentam características dos problemas reais, como por exemplo mudança de elevação de terrenos.

2.7 Funções *Benchmark*

O uso das funções *benchmark* é um ponto crucial para o desenvolvimento, comparação, avaliação e otimização dessa classe de problemas (NGUYEN; YANG; BRANKE, 2012). O fato das condições do problema variarem de acordo com o tempo acrescenta uma camada de dificuldade a mais para obter uma solução satisfatória após uma mudança. Um bom *benchmark* deve possuir as seguintes características.

1. Flexibilidade: Configurável sobre diferentes abordagens dinâmicas, como por exemplo, a frequência ou intensidade das alterações no ambiente, em diferentes escalas.

2. Simplicidade e Eficiência: Deve ser fácil de implementar e analisar.

3. Associação com Problemas Reais: Ter a capacidade de fazer previsões em problemas reais ou se assemelhar em alguma extensão.

Para a analisar um *benchmark* é necessário dar atenção para o tipo de dinamismo que esse problema apresenta, pois para otimizá-lo são escolhidas estratégias de otimização baseadas no tipo de dinamismo. Cada *benchmark* possui suas características e com isso cada algoritmo aplicado a ele terá que analisar como está abordando essas peculiaridades. As Características observadas nas funções encontradas são (CRUZ; GONZÁLEZ; PELTA, 2011):

1. Influência Temporal: Sempre que uma solução futura depende de outra encontrada anteriormente pelo algoritmo, ou seja, quando o tipo de mudança que o algoritmo vai sofrer depende de quais soluções foram encontradas ao decorrer da otimização. Neste caso os resultados podem mudar para cada tentativa de otimização.

2. Previsibilidade: Quando as mudanças geradas pelo problema seguem um padrão que pode ser previsto durante a otimização do problema. Podem ser alterações por mudança em uma escala fixa, incremental, cíclica ou periódica.

3. Visibilidade: Sempre que as mudanças do ambiente são visíveis pelo algoritmo otimizando o problema. O algoritmo não deve precisar utilizar muitos detectores para perceber a mudança no ambiente, podem ser pontos específicos do espaço, que são detectados ao reavaliar a função objetivo, ou até mesmo, as restrições.

4. Problemas com Restrições: As mudanças do problema podem ocorrer nas restrições, que podem mudar ao longo do tempo (a variável de tempo influencia na função de restrição) ou podem ser ativadas e/ou desativadas durante a otimização.

5. Número de Objetivos: São os problemas multiobjetivos, que neste caso podem alterar o número de objetivos durante a otimização, chegando até, ao ponto de ter somente um objetivo em vista.

6. Intensidade das Mudanças: Indica qual a escala da mudança, ou seja, quão expressiva é está alteração em relação ao *fitness* do indivíduo, tendo como preocupação o limite das mudanças.

7. Fator da Mudança: Onde o dinamismo pode ocorrer, sendo na função objetivo, no domínio das variáveis no número de variáveis, nas restrições ou outros parâmetros.

2.7.1 Avaliação de Desempenho

Para avaliar um *benchmark* são estipuladas métricas que serão utilizadas para medir a qualidade dos resultados obtidos pelo algoritmo de otimização. Os métodos de avaliação apresentados foram selecionados por serem amplamente utilizados em algoritmos bioinspirados aplicados em ambientes dinâmicos e serão usados para avaliar o desempenho dos algoritmos selecionados neste trabalho. A forma de avaliação pode ser dividida em dois grandes grupos: Medida de desempenho baseados em otimalidade e em comportamento.

Otimalidade

As medidas de desempenho baseadas na otimalidade avaliam a capacidade dos algoritmos em encontrar as soluções com o melhor valor de *fitness* ou as que estão mais próximas

do ótimo global (medidas baseadas na distância) e essa avaliação leva em conta o algoritmo como um todo. Entre elas tem-se:

- Melhor da Geração: Avalia qual o melhor indivíduo de cada iteração do algoritmo, gerando assim uma curva de desempenho que apresenta os pontos de melhora do algoritmo e o quão perto está do ótimo do problema. Esse é o método mais frequentemente utilizado.
- Melhor Erro Antes da Mudança: Calcula a média do erro (diferença da melhor solução atual para o ótimo do problema) da população logo antes da mudança do ambiente. Este caso só pode ser usado quando o dinamismo do *benchmark* gerado é previsível ou facilmente identificável, que por sua vez, dependo de como o *benchmark* muda o ambiente.
- Pontuação Normalizada: Mede a diferença entre uma determinada gama de algoritmos em relação a uma lista predeterminada de problemas, fazendo com que cada algoritmo execute cada problema um número determinado de vezes. Com os resultados normalizados, para a melhor solução é atribuído o valor 1 (um) e para a pior solução é atribuído o valor 0 (zero).

Para quantificar a otimalidade é utilizado o *Offline Error* (OFFE) (BRANKE; SCHMECK, 2003), que representa o quanto o algoritmo chegou perto do ponto ótimo. Essa métrica é usada pois nos *benchmarks* os valores de *fitness* gerados, são diferentes para cada execução, sendo assim, com o valor de *Offline Error* pode-se estipular uma aproximação para cada execução em média. O cálculo dessa métrica é feito pela diferença do melhor da população.

$$OFFE = fitness_{best} - population_{best} \tag{2.9}$$

Onde o $fitness_{best}$ é o objetivo da função avaliada e o $population_{best}$ é o melhor resultado encontrado pela população até agora.

Comportamento

As medidas de desempenho baseadas em comportamento avaliam a influência de certos comportamentos que são importantes na hora de manter um nível de *fitness*, como por exemplo:

- Manutenção da Diversidade: Como o nome já diz, mede quanto que um comportamento pode ajudar na manutenção da diversidade da população ao longo do processo de otimização. Este é um fator muito estudado em ambientes dinâmicos, pois com uma grande diversidade da população existem mais chances de se encontrarem os novos ótimos globais.

- Velocidade de Convergência Após uma Mudança: Mede quão rápido o sistema se adapta após perceber uma alteração. Essa medida de comportamento não se aplica a sistemas que tem dificuldades em perceber uma alteração no ambiente. Para analisar essa medida é feita uma análise de quantas iterações são necessárias para se encontrar um novo ótimo, em relação a distância.

- Performance Após uma Mudança: Mede a capacidade do algoritmo de buscar novos resultados após as mudanças no ambiente, de forma que restringe a queda do *fitness* quando percebe uma piora. Essa medida também é conhecida como medida de estabilidade ou satisfatibilidade.

2.8 Instâncias de Problemas

Nesta etapa são apresentados e detalhados os problemas que serão abordados no trabalho. Os problemas selecionados para esse trabalho serão usados para realizar uma análise comparativa com outras abordagens encontradas na literatura e também testar o desempenho de cada operador evolutivo que os algoritmos irão abordar. A maioria dos problemas selecionados não possui um *link* temporal com as soluções anteriores. São apresentados com e sem restrições na função objetivo ou no limite do domínio das variáveis, tendo em sua maioria dinamismos previsíveis, periódicos e constantes.

2.8.1 *Moving Peaks* - MP

O *benchmark* de movimentação de picos (*moving peaks*) foi desenvolvido por Jürgen Branke (BRANKE, 1999) e tem como principal ideia gerar um problema que possa ser amplamente utilizado em algoritmos bioinspirados. A característica diferencial do MP em relação a maioria dos problemas dinâmicos, até o dado momento, é que eles sofriam uma alteração drástica que gerava um novo ambiente totalmente diferente do anterior, tornando assim um

recomeço da otimização uma solução melhor.

A ideia é gerar um *benchmark* que, ao sofrer uma mudança no ambiente, ficasse levemente alterado, o suficiente para que as soluções anteriores ajudassem a encontrar as novas geradas. Para isso foi criado o *moving peaks* com a ideia de ter uma paisagem multidimensional artificial que consiste em vários picos, em que a altura, a largura e a posição de cada pico é ligeiramente alterada cada vez que uma mudança no ambiente ocorre.

A função objetivo é dada pela Equação 2.10, na qual tem-se H como altura, W como peso, n como o número de picos e m como o número de dimensões. Todos esses parâmetros são definidos previamente aos testes.

$$F(\vec{x}, t) = \max_{i=1 \to n} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{m} (x_j - X_j(t))^2} \tag{2.10}$$

A cada Δe (valor que representa o intervalo/frequência das mudanças do ambiente), a altura e o peso são alterador adicionando uma variável aleatória. O local de cada um dos picos é movido por um vetor $\vec{v}$ a uma distância s fixa para uma direção aleatória. Essas alterações podem ser descritas pela Equação 2.11.

$$\begin{aligned} &\sigma \in N(0,1) \\ &H_i(t) = H_i(t-1) + 7.\sigma \\ &W_i(t) = W_i(t-1) + 0,01.\sigma \\ &\vec{X}_i(t) = \vec{X}_i(t-1) + \vec{v} \end{aligned} \tag{2.11}$$

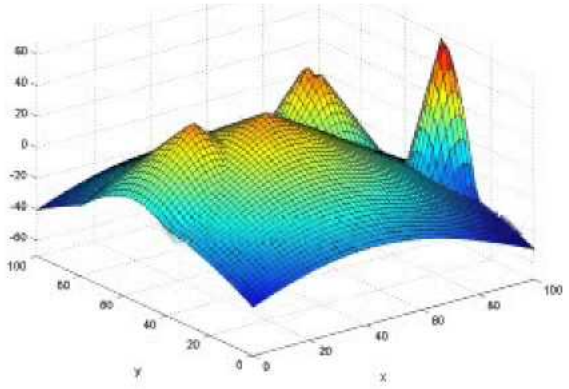
Um exemplo de como as máximas se movem ao longo do tempo em um espaço bidimensional pode ser visto na Figura 2.3

2.8.2 *Ocillating Peaks* - OP

O *benchmark* de oscilação de picos (*ocillating peaks*) é baseado no de movimentação de picos, porém foi desenvolvido para algoritmos evolutivos com utilização de memória.

É uma combinação linear entre duas funções em que o peso delas se alteram ao longo do tempo, portanto, a função G, começa em g_1, passa para g_2, depois volta para g_1 e assim sucessivamente. Em outras palavras, o máximo absoluto de G oscila entre dois pontos

Figura 2.3: Gráfico de duas dimensões do movimento dos picos, após 300 alterações no ambiente, $s = 0,9$



Fonte: Branke (1999).

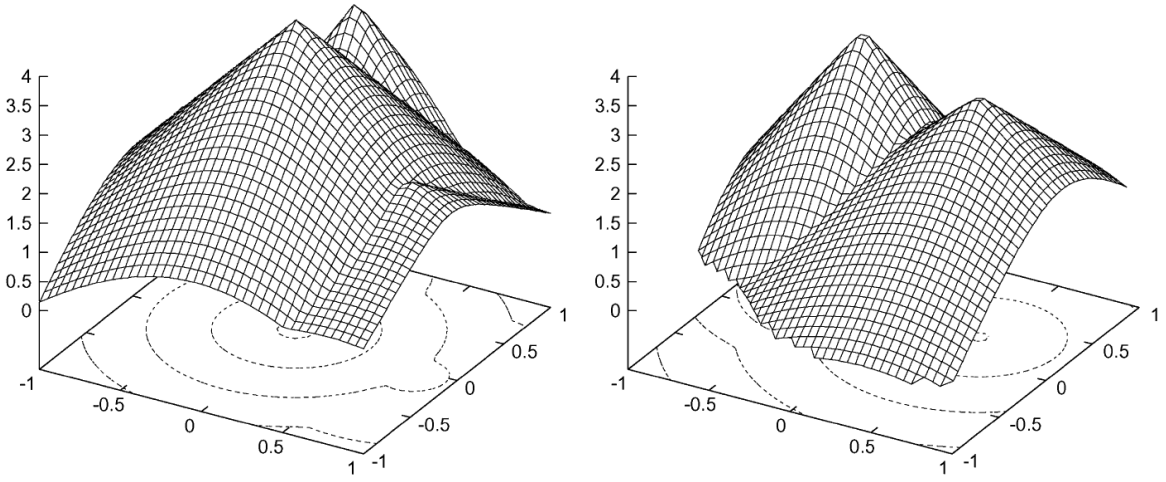
g_1 e g_2. As funções que representam as alterações da função G estão representadas na Equação 2.12, em que, $\lambda(t)$ representa a oscilação da função, n é a soma dos picos das duas funções que compõem a combinação linear e m é o número de dimensões.

$$
\begin{aligned}
&\delta \in N(0,1) \\
&\lambda(t) = \frac{\cos\frac{2\pi t}{100\delta} + 1}{2} \\
&g_1(\vec{x}) = \sum_{i=1}^{n/2} \frac{H_i}{1 + W_i \sum_{j=1}^{m}(x_j - X_j(t))^2} \\
&g_2(\vec{x}) = \sum_{i=n/2+1}^{n} \frac{H_i}{1 + W_i \sum_{j=1)}^{m}(x_j - X_j(t))^2} \\
&G(t,\vec{x}) = \lambda(t) g_1(\vec{x}) + (1 - \lambda(t)) g_2(\vec{x})
\end{aligned}
\tag{2.12}
$$

2.8.3 Gerador de Problemas de teste para Ambientes não Estacionários

O Gerador de Problemas de testes para Ambientes não Estacionários (DF1) foi utilizado para gerar ambientes, tais como o mostrado na Figura 2.4 proposto por Morrison e Jong (MORRISON; DE JONG, 1999). Este gerador é capaz de criar um número especificado de pontos ótimos em duas dimensões utilizando a Equação 2.13

Figura 2.4: Gráfico que representa os ambientes gerados pelo Gerador de Problemas testes para Ambientes não Estacionários



Fonte: Morrison e De Jong (1999).

$$
\begin{aligned}
& f(X,Y) = \max_{i=1,N} \left[H_i - R_i . \sqrt{(X - X_i)^2 + (Y - Y_I)^2} \right] \\
& H_i \in [H_{base}, H_{base} + H_{range}] \\
& R_i \in [R_{base}, R_{base} + R_{range}]
\end{aligned}
\tag{2.13}
$$

Em que N representa o número de pontos ótimos a serem gerados, (X, Y) representa sua posição, H_i representa sua altura e R_i representa sua inclinação. o limite da altura e inclinação são apresentados na Equação 2.13. Nesse sistema variam suas posições, formas e altura. A altura varia entre um valor máximo e mínimo, criando assim um perfil de dente de serra, enquanto a altura é representada graficamente. O *fitness* de cada ponto da superfície é atribuído a altura máxima de todos os pontos ótimos.

Embora o número de pontos ótimos presente não possa ser alterado uma vez que o ambiente tenha sido inicializado, a combinação da mudança altura e posições faz com uma solução ótima fique temporariamente obscura em relação aos vizinhos. Esse fenômeno gera a impressão que alguns pontos desaparecem e ressurgem, porém a quantidade de picos não é alterada. O dinamismo do ambiente pode ser especificado por um parâmetro A que varia de [0, 4]. A direção de cada etapa é escolhida aleatoriamente para coordenadas espaciais, com paços que podem colocar um pico fora dos limites das variáveis. A direção da mudança nas variáveis de altura e inclinação são escolhidas aleatoriamente inicialmente e continuam nessa direção até exceder o intervalo.

3 Trabalhos Relacionados

A Computação Natural possui vários operadores diferentes em algoritmos diferentes para serem aplicados em diversos tipos de problemas. Neste capítulo são apresentados os trabalhos do estado da arte encontrados na literatura que utilizam esses algoritmos bioinspirados em problemas dinâmicos de domínio contínuo.

3.1 Evolução Diferencial Local a Base de Aglomeração e com Memória Baseada em Espécies

No trabalho de (KUNDU, 2013), é feito uma análise comparativa das versões do DE existentes, especificando suas necessidades ao serem aplicados em ambiente dinâmicos e ao final é apresentado a proposta de aplicação. As versões que são analisadas são as seguintes:

- Evolução Diferencial Baseada em Aglomeração (*Crowding-based DE* - CDE) (THOMSEN, 2004): Basicamente, o CDE estende DE com o esquema *Crowding*. Assim, a única modificação ao DE convencional é sobre o indivíduo (pai) ser substituído. Normalmente o pai que produz a descendência é substituído, ao passo que no CDE a descendência substitui o indivíduo mais similar entre um subconjunto da população.

- Evolução Diferencial Baseada em Espécies (*Species-based DE* - SDE) (LI, 2005): Seguindo os passos para determinar sementes de espécies (centros de populações menores dentro da população original), o SDE é capaz de identificar múltiplas espécies entre toda a população a cada iteração. Cada espécie identificada é otimizada por uma instância do DE.

- Evolução Diferencial Baseada em Compartilhamento (*Sharing-based DE* - ShDE) (THOMSEN, 2004): O SDE modifica o DE convencional da maneira seguinte. Em primeiro lugar, em vez dos pais serem substituídos por todas as crias, elas são adicionados à população. Em segundo lugar, a adequação de todos os indivíduos é redimensionada usando a função de *Sharing*. Em terceiro lugar, a população é ordenada no que diz respeito ao novo *fitness*. Finalmente, a pior metade da população (igual ao tamanho da

população inicial) é removido.

Após a análise dos algoritmos é demonstrado a proposta do autor, em que pode-se ver os pontos que foram adaptados dos trabalhos citados anteriormente e quais aspectos desses algoritmos foram alterados. No novo modelo, chamado de Evolução Diferencial Local à Base de Aglomeração e com Memória Baseada em Espécies (*Crowding-based local Differential Evolution with Speciation-based Memory* - ClDES) existem 3 características que se diferenciam do DE convencional, sendo elas:

1. Mutação por Vizinhança: durante a mutação feita pelo ClDES são escolhidos somente os indivíduos que estão próximos a vizinhança, de forma que, em relação a população total, somente uma pequena parte pode ser escolhida.

2. Função de Teste: É estipulada uma função de teste para detectar mudanças no ambiente que não faz parte do processo evolutivo, porém a cada geração essa função é testada e se for identificado alguma alteração no resultado do *fitness* fica comprovada a mudança no ambiente. Ao ser identificada essa mudança, o algoritmo toma as ações necessárias para manter um determinado nível de *fitness*.

3. Arquivo de Memória Baseada em Especiação: ao detectar uma mudança no ambiente, o sistema procura os núcleos de espécies, chamados de sementes de espécies, e a partir dele são selecionados indivíduos que serão mantidos na nova população. A nova população feita utilizando metade dos indivíduos selecionados pelas sementes de espécies e a outra metade é gerada aleatoriamente.

O ClDES foi aplicado no MP *benchmark* e foi comparado com outras versões do PSO, que são: *Dynamic Species-Based Particle Swarm Optimizer* (DSPSO) e *Clustering Particle Swarm Optimizer* (CPSO), em que mostra-se superior a ambas as versões em quase todos os casos. A principal dificuldade encontrada pelo algoritmo acontece quando o número de dimensões aumenta, tornando cada vez mais difícil do algoritmo sempre encontrar todos os picos do problema.

3.2 Algoritmo de Vaga-Lumes baseado em multi-enxames - MSFA

O trabalho de (FARAHANI; NASIRI; MEYBODI, 2011) propõe uma junção do FA com o método de multi-enxames (*MultiSwarm* - MS). A ideia principal da técnica MS é dividir a população em um número de sub enxames, com o objetivo de posicionar cada um desses sub enxames sobre diferentes áreas, visando encontrar diferentes picos no espaço de busca. No entanto, simplesmente separar o enxame em um número de enxames independentes não são susceptíveis de serem eficazes, uma vez que não tenha interação entre os sub enxames. Existem algumas abordagens chamadas de Exclusão e Anti-convergência para resolver este problema.

1. Exclusão: A exclusão é uma interação local entre enxames próximos de colidir. Se um enxame é dividido em um certo número de sub enxames, pode acontecer que partículas de diferentes enxames girem em torno de um único pico. Isso é indesejável uma vez que a motivação por trás de uma abordagem MS é postular diferentes enxames em diferentes picos. A fim de evitar isso é feito uma competição entre os enxames, quem possuir o melhor *fitness* continua na otimização, o outro sub enxame é extinto e é gerado novamente.

2. Anti-convergência: Anti- convergência é uma partilha de informações de cada interação entre todos os sub enxames como uma interação global no algoritmo MS, com o objetivo de permitir que novos picos sejam detectados.

O algoritmo proposto usa a MS para localizar todos o picos do MP *benchmark*, em que cada sub enxame é um FA. Para a manutenção da diversidade no sistema MS utiliza-se partículas carregadas e partículas quânticas. Devido à lenta velocidade de convergência do FA e as armadilhas em vários locais ótimos do espaço de busca, neste trabalho, um novo comportamento é introduzido que melhora o desempenho do FA. No algoritmo proposto todas as partículas quânticas de cada enxame irão se mover em direção do melhor vaga-lume global (G_best) se não houver nenhum vaga-lume melhor entre seus seus vizinhos. Este comportamento melhora a velocidade de convergência. Também para cobrir eventuais desvios no movimento de vaga-lume, é usado um ângulo que torna o movimento dos vaga-lumes previsíveis e dá uma direção para cada vaga-lume.

Os resultados obtidos da aplicação do Algoritmo de Vaga-Lumes Baseado em Multi-Enxames (*Multiswarm Based Firefly Algorithm* - MSFA) no MP *benchmark* mostram

que o MSFA mostra-se superior ao FA e a outras versões do PSO. A principal característica constatada é quando o número de picos é diferente do número de sub enxames, de forma que se o número de sub enxames for menor o desempenho do algoritmo cai drasticamente.

3.3 Algoritmo PSO em Ambientes Dinâmicos

O PSO é um dos algoritmos de SI mais estudados na área de otimização de problemas dinâmicos, de forma que são geradas várias versões dele com vários operadores interessantes (CARLISLE, 2002).

3.3.1 *Dynamic Species-Based Particle Swarm Optimizer* - DSPSO

O DSPSO (PARROTT; LI, 2006) é baseado na sua versão estacionária, o SPSO que por sua vez é baseado no sistema de espécies. Um sistema baseado em espécies possui vários nichos de espécies e em cada um deles existe um centro geográfico chamado de semente da espécie (*specie seed*). Esses nichos de espécies são utilizados para que o sistema possa encontrar vários pontos ótimos durante o processo de otimização, fazendo com o que sistema não fique estagnado em um pico subótimo.

Para encontrar os pontos ótimos após uma mudança no ambiente, para cada partícula da população é reavaliado o *fitness* em função do melhor *fitness* da população. Então cada partícula usa sua informação de aptidão para realizar o movimento, tendo sempre uma noção prévia do ambiente ao seu redor. Para incentivar o enxame a realizar uma busca no ambiente e evitar uma convergência prematura do sistema, foi introduzido o parâmetro de limite de partículas por espécies (P_{max}), de forma que se uma partícula fosse entrar para um espécie mas o limite de integrantes daquela espécie exceder, a partícula com o menor valor de *fitness* é reiniciada. Uma desvantagem desse sistema é que gera uma lentidão no processo de convergência se o P_{max} for muito baixo.

Nos resultados pôde-se observar que nem sempre o DSPSO obteve resultados mais precisos, porém sempre foi capaz de encontrar mais ou o mesmo número de picos que os outros algoritmos.

3.3.2 *Clustering Particle Swarm Optimizer* - ClPSO

Outra versão do PSO é o *Clustering Particle Swarm Optimizer* ClPSO (YANG; LI, 2010), em que também utiliza o modelo de multienxames explicado na Seção 3.2.

ClPSO começa a partir de um enxame inicial, chamado de enxame berço. Em seguida, subenxames são criados por um método de clusterização hierárquico. Quando subenxames são criados, uma busca local é aplicada sobre eles, a fim de explorar potenciais picos próximos deles. Após isso é feita uma checagem de subenxames sobrepostos, convergência prematura, e superlotação antes dos próximos ciclos de iteração. Se uma mudança ambiental é detectada, um novo enxame berço será aleatoriamente regerado com a reserva das posições dos sobreviveram dos subenxames anteriores.

A divisão do enxame berço para os subenxames é feita por um processo de clusterização hierárquico de ligação única. O processo de busca local utiliza a partícula de melhor *fitness* global para direcionar o enxame berço, a fim de obter uma convergência mais rápida. Esse modelo de busca local é o utilizado na versão normal do PSO.

O resultados do algoritmo aplicados no *benchmark* MP mostram que ele teve um performance superior as outras versões do PSO analisadas, de forma que quase sempre foi possível encontrar todos os picos e o resultado ótimo.

3.3.3 *Volitive Particle Swarm Optimizer* - VPSO

Uma aplicação do PSO gerou uma versão híbrida dele com o algoritmo de busca por cardume de peixes (*Fish School Search Algorithm* FSS) (CARMELO FILHO, 2008), utilizando os operadores de alimentação e de movimentação coletiva volátil. No *Volitive Particle Swarm Optimizer* (VPSO) (CAVALCANTI-JÚNIOR, 2011), cada partícula torna-se um indivíduo ponderado, em que o peso é usado para calcular o movimento coletivo volátil, resultando na expansão ou contração do enxame. Nessa proposta o $step_{vol}$ não diminui linearmente, ele decresce a partir da Equação 3.1. A percentagem de decaimento do parâmetro volátil ($decay_{vol}$) deve estar no intervalo [0,100].

$$step_{vol}(t+1) = step_{vol}(t)\frac{100 - decay_{vol}}{100} \tag{3.1}$$

O $step_{vol}$ é reinicializado para o $step_{volMax}$ quando uma mudança no ambiente é detectada. É

utilizada uma partícula sentinela para detectar estas mudanças. A aptidão da partícula sentinela é avaliada no final de cada iteração e no início da iteração seguinte. Este Algoritmo foi aplicado no DF1 *benchmark* e obteve resultados melhores que suas versões anteriores, mostrando assim a grande influência que o operador de movimento volátil tem sobre os ambientes dinâmicos.

3.4 Otimização por Colônia de Bactérias em Problemas Dinâmicos

O processo de reprodução da BFA que tem o objetivo de acelerar a convergência é adequado em problemas estáticos, porém isso gera uma falta de adaptação em ambientes dinâmicos. Assim, a fim de se comprometer entre a convergência rápida e a alta diversidade, é proposto um algoritmo dinâmico de Otimização por Colônia de Bactérias (*Dynamic Bacterial Foraging Algorithm* - DBFA) (PASSINO, 2002) em que não é utilizada a eliminação/dispersão. Um processo de seleção é introduzido através de um esquema mais flexível para permitir uma melhor capacidade de adaptação em um ambiente em uma mudança. A ideia básica do DBFA é manter uma diversidade adequada para pesquisa global, enquanto a capacidade de busca local não é degradada, e também considerar as alterações no ambiente. O processo de seleção é descrito na Equação 3.2

$$\begin{aligned} J_i &= \sum_{j=1}^{n} J_i(j,r) \\ rank_i &= sort(J_i) \\ W_i &= m\frac{(rank_i)^k}{\sum_{i=1}^{P}(rank_i)^k} + (1-m)\frac{J_i}{\sum_{i=1}^{P} J_i} \end{aligned} \tag{3.2}$$

em que n é o número de etapas quimiotáticos (cada passo pode conter uma corrida ou um tombo) durante o tempo de vida de uma bactéria, j é o seu índice e P é o tamanho da população, o símbolo m representa o peso de diversidade, e k é o expoente de classificação $rank_i$. Assim, essas áreas mais ricas em nutrientes experientes (melhor *fitness*) são mais propensas a serem selecionadas como um pai para a próxima geração. No entanto, esse domínio não ajudaria na manutenção da diversidade.

Em seguida tem-se a combinação da solução e da classificação, que são escolhidas para evitar uma rápida convergência e devem ser evitadas para manter uma capacidade de

adaptação do DBFA. Assim, toda a população é classificada de acordo com J_i usando um tipo de operador, em seguida, $rank_i$ é alocado como classificação da bactéria i. É utilizado o parâmetro m que afeta a diversidade para o processo de seleção através da combinação da classificação da bactéria ($rank_i$) k com o cálculo de *fitness* J_i. A probabilidade de sobrevivência da bactéria é determinada pela somatória da variável W_i. Ao final é aplicada a seleção por roleta utilizada nos AGs.

O BFA e o DBFA foram aplicados no *benchmark* MP e foram avaliados seus desempenho e sua diversidade durante a otimização. Foi constatado que a diversidade de DBFA muda depois de cada processo quimiotático em vez da dispersão adotada pelo BFA, que ocorre depois de várias gerações. O DBFA utiliza não só a busca local, mas também aplica o esquema de seleção flexível para manter uma diversidade apropriada durante todo o processo evolutivo. O DBFA supera o BFA em quase todos os ambientes dinâmicos. Além disso, a detecção de mudanças ambientais não é necessária no DBFA. O DBFA tem a mesma complexidade computacional que a do BFA , mas oferece o melhor desempenho.

3.5 Considerações

Com a análise destes problemas foi possível extrair algumas informações relevantes, que estão esquematizadas na Tabela 3.1, entre elas tem-se os problemas abordados pelos algoritmos, suas rotinas de manutenção da diversidade populacional e por final é apresentado algumas outras informações importantes de cada problema.

O CIDES utiliza como conceito as suas versões anteriores, para que assim pode-se diminuir a quantidade de erro da sua aplicação em problemas dinâmicos. A sua mutação por vizinhança ajudou a manter a busca global mais controlada, de forma que pudesse buscar por valores mais próximos de forma aleatória. Sua função de teste é uma das formas mais básicas estudadas de perceber mudanças no ambiente, o que é uma ferramenta muito importante na aplicação dos algoritmos em ambiente que sofrem essas mudanças constantemente.

O MSFA mostra a importância da aplicação da abordagem de multienxames, que tem como base a coevolução. Suas rotinas de exclusão e anti convergência formam a comunicação entre as pequenas populações que existem na otimização, abstraindo a otimização um nível a cima de apenas uma população.

O VPSO demonstra uma aplicação do operador de movimentação coletiva volátil

Quadro 3.1: Tabela das Relações dos Algoritmos Analisados

Algoritmos	Problemas Abordados	Rotinas de manutenção da diversidade
CDE	MP *benchmark*	*Crowding*
SDE	OP *benchmark*	Diferenciação por espécies
ShDE	MP *benchmark*	*Sharing*
ClDES	MP *benchmark*	Mutação por Vizinhança
MSFA	MP *benchmark*	Partículas carregadas e Quânticas
DSPSO	MP *benchmark*	Limitação de partículas por Espécies
CPSO	MP *benchmark*	Clusterização Hierárquico de Ligação Única
VPSO	DF1 *benchmark*	Movimentação volátil
DBFA	MP *benchmark*	Combinação da Classificação das Bactérias

do FSS, que é o foco da pesquisa deste trabalho, e mostra que tem grandes possibilidade para ajudar a manter a população no caminho certo durante as mudanças do ambiente.

No DBFA pode-se notar mais uma versão conjunta de dois algoritmos, em que usa-se o Algoritmo BFA com a rotina de seleção do AG. Isso mostra que mesmo algoritmos de classes diferentes (como os EA com o AG e os SI com o BFA) podem ser aplicados em conjunto e terem resultados melhores que separados.

Em geral a manutenção da diversidade da população sempre tem sido um dos focos no desenvolvimento de novas abordagens de algoritmos bioinspirados, com isso, pode-se afirmar que nessa classe de problemas esse é um ponto crucial que deve ser levado em consideração e analisado com técnicas diferentes para estipular a relevância de cada uma.

4 Modelo

Neste capítulo, uma nova versão baseada no Algoritmo de Otimização Enxame de Partículas em Clãs, chamado de Algoritmo de Otimização por Enxame de Partículas em Clãs Dinâmico (*Dynamic Clan Particle Swarm Optimization*, DCPSO), será apresentada para utilização na otimização de problemas dinâmicos de domínio contínuo.

Essa nova versão do CPSO utiliza uma rotina de aglomeramento (*crowding*) como manutenção da diversidade e uma rotina nova chamada de explosão, que consiste em um reinicialização de alguns clans quando a diversidade entre esses clans forem muito baixa e o ambiente sofrer uma mudança.

4.1 Características do algoritmo

Inicialmente o CPSO, sem nenhuma alteração, aplicado a problemas dinâmicos não teve uma performance muito boa em manter o rastreamento de vários picos ao mesmo tempo. Então foram definidos modificação para ajudar neste processo de rastreamento e assim na melhoria da performace do algoritmo. Algo que foi adicionado também foi uma rotina de verificação de mudança, pois as rotinas citada anteriomente são ações reativas e precisam saber quando acontece uma mudança no ambiente

1. Função de Aglomeramento (*crowding*): Tem como abjetivo ajudar os clãs a se manterem separados e assim a convergência acontece melhor dentro de cada um dos clãs.

2. Função de Explosão: Tem como objetivo reinicializar os clãs que se aglomeraram, assim permitindo que o clã resetado procure outro pico (ótimo local) para fazer o rastreamento e assim almentado a diversidade populacional e entre clãs.

3. Função de Detecção de Mudança: É executada toda iteração e tem como objetivo detectar se o ambiente sofreu alguma mudança. Para isso foi criada uma partícula de teste que acrescenta uma avaliaçào de fitness a mais para cada execução.

4.1.1 Função de Aglomeramento (*crowding*)

4.2 Ilustração Conceitual

Nesta seção será apresentado um modelo de ilustração conceitual para demotrar a execução do algoritmo em um ambiente de teste com duas dimenções

5 Protocolo de Experimentação

6 Resultados e Análises

7 Conclusão e Trabalhos Futuros

De um modo geral, a maioria das pesquisas em algoritmos evolutivos e de enxame concentram-se em problemas de otimização estáticos. No entanto, muitos problemas do mundo real são problemas de otimização dinâmica, em que as mudanças ocorrem ao longo do tempo. Isso requer algoritmos de otimização, não só para encontrar a solução ideal global sob um ambiente específico, mas também para monitorar continuamente a mudança em diferentes ambientes dinâmicos. Assim, são necessários métodos de otimização que são capazes de se adaptar continuamente.

Os algoritmos de inspiração biológica são amplamente estudados para melhorar sua desempenho em ambientes dinâmicos e com isso vários operadores são gerados e aplicados em diferentes problemas dinâmicos com diferentes propriedades. Cada um desses operadores possuem pontos fortes e fracos nas suas aplicações, então o estudo comparativo para identificar esse pontos se faz necessário, de modo que a aplicação em conjunto desses operadores pode-se benéfico para ambos, neutralizando esses pontos negativos.

O trabalho até aqui contribui exatamente nesse estudo comparativo das técnicas de otimização em ambientes dinâmicos. Nos estudos feitos até agora pode-se notar uma grande quantidade de operadores evolutivos e suas influências, de modo que a aplicação desses operadores no FSS pode ser esquematizada para que, na próxima etapa do trabalho, seja concluída. Os métodos de manutenção da diversidade populacional contribuem em geral na otimização dos ambientes dinâmicos com domínio contínuo pelo fato de que uma alta diversidade contribui na manutenção do nível de *fitness*.

Dado o desenvolvimento do Trabalho de Conclusão de Curso até o momento, é apresentado as próximas etapas a serem desenvolvidas pelo cronograma até o final do trabalho, listados abaixo e ilustradas na Tabela 7.1.

1. Analisar a influência dos operadores encontrados e avaliar quais devem ser aplicados no algoritmo;

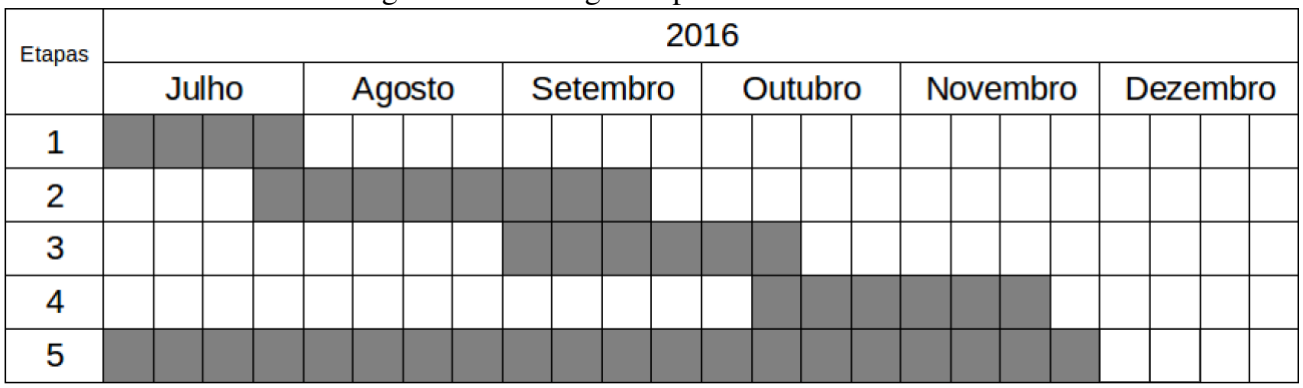
2. Desenvolver a versão definitiva do algoritmo utilizando todos os operadores evolutivos alternadamente;

3. Experimentos computacionais com o algoritmo e os problemas selecionados;

4. Coleta e análise dos resultados dos experimentos;

5. Finalização da escrita do trabalho de conclusão de curso

Figura 7.1: Cronograma para o TCC - 2



Fonte: Produção do próprio autor.

Bibliografia

ANDRÉ, L.; STUBS PARPINELLI, R. The multiple knapsack problem approached by a binary differential evolution algorithm with adaptive parameters. *Polibits*, Instituto Politécnico Nacional, Centro de Innovación y Desarrollo Tecnológico en Cómputo, n. 51, p. 47–54, 2015.

BELLMAN, R. E.; DREYFUS, S. E. *Applied dynamic programming*. : Princeton university press, 2015. 200–206 p.

BOOKER, L. B.; GOLDBERG, D. E.; HOLLAND, J. H. Classifier systems and genetic algorithms. *Artificial intelligence*, Elsevier, v. 40, n. 1, p. 235–282, 1989.

BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. *Information Sciences*, Elsevier, v. 237, p. 82–117, 2013.

BRANKE, J. Memory enhanced evolutionary algorithms for changing optimization problems. In: CITESEER. *In Congress on Evolutionary Computation CEC99*. 1999.

BRANKE, J. *Evolutionary optimization in dynamic environments*. : Springer Science & Business Media, 2012.

BRANKE, J.; SCHMECK, H. Designing evolutionary algorithms for dynamic optimization problems. *Advances in evolutionary computing*, Springer, p. 239–262, 2003.

CARLISLE, A. J. *Applying the particle swarm optimizer to non-stationary environments*. : Auburn University, 2002.

CARMELO FILHO, J. et al. A novel search algorithm based on fish school behavior. In: IEEE. *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*. 2008. p. 2646–2651.

CARVALHO, D. Ferreira de; JOSÉ ALBANEZ BASTOS-FILHO, C. Clan particle swarm optimization. *International Journal of Intelligent Computing and Cybernetics*, Emerald Group Publishing Limited, v. 2, n. 2, p. 197–227, 2009.

CASTRO, L. N. de. Fundamentals of natural computing: an overview. *Physics of Life Reviews*, Elsevier, v. 4, n. 1, p. 1–36, 2007.

CAVALCANTI-JÚNIOR, G. M. et al. A hybrid algorithm based on fish school search and particle swarm optimization for dynamic problems. In: *Advances in Swarm Intelligence*. : Springer, 2011. p. 543–552.

CORRIVEAU, G. et al. Review and study of genotypic diversity measures for real-coded representations. *Evolutionary Computation, IEEE Transactions on*, IEEE, v. 16, n. 5, p. 695–710, 2012.

CORRIVEAU, G. et al. Review of phenotypic diversity formulations for diagnostic tool. *Applied Soft Computing*, Elsevier, v. 13, n. 1, p. 9–26, 2013.

ČREPINŠEK, M.; LIU, S.-H.; MERNIK, M. Exploration and exploitation in evolutionary algorithms: a survey. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 3, p. 35, 2013.

CRUZ, C.; GONZÁLEZ, J. R.; PELTA, D. A. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, Springer, v. 15, n. 7, p. 1427–1448, 2011.

DE JONG, K. A. Analysis of the behavior of a class of genetic adaptive systems. 1975.

EBERHART, R. C.; KENNEDY, J. et al. A new optimizer using particle swarm theory. In: NEW YORK, NY. *Proceedings of the sixth international symposium on micro machine and human science*. 1995. v. 1, p. 39–43.

FARAHANI, S. M.; NASIRI, B.; MEYBODI, M. R. A multiswarm based firefly algorithm in dynamic environments. In: CITESEER. *Third Int. Conf. on Signal Processing Systems (ICSPS2011)*. 2011. v. 3, p. 68–72.

GARNIER, S.; GAUTRAIS, J.; THERAULAZ, G. The biological principles of swarm intelligence. *Swarm Intelligence*, Springer, v. 1, n. 1, p. 3–31, 2007.

GOLDBERG, D. E.; RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In: HILLSDALE, NJ: LAWRENCE ERLBAUM. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*. 1987. p. 41–49.

HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* : U Michigan Press, 1975.

KUNDU, S. et al. Crowding-based local differential evolution with speciation-based memory archive for dynamic multimodal optimization. In: ACM. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013. p. 33–40.

LI, C.; YANG, M.; KANG, L. A new approach to solving dynamic traveling salesman problems. In: *Simulated Evolution and Learning*. : Springer, 2006. p. 236–243.

LI, X. Efficient differential evolution using speciation for multimodal function optimization. In: ACM. *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005. p. 873–880.

MAHFOUD, S. W. Niching methods for genetic algorithms. *Urbana*, Citeseer, v. 51, n. 95001, p. 62–94, 1995.

MENGSHOEL, O. J.; GOLDBERG, D. E. Probabilistic crowding: Deterministic crowding with probabilistic replacement. In: *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-99)*. 1999. p. 409.

MORRISON, R. W. Performance measurement in dynamic environments. In: *GECCO workshop on evolutionary algorithms for dynamic optimization problems*. : Citeseer, 2003. p. 5–8.

MORRISON, R. W.; DE JONG, K. A. A test problem generator for non-stationary environments. In: IEEE. *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. 1999. v. 3.

MOSER, C. I. Review all currently known publications on approaches which solve the moving peaks problem. Citeseer, 2007.

NGUYEN, T. T.; YANG, S.; BRANKE, J. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, Elsevier, v. 6, p. 1–24, 2012.

OLIVEIRA, M. C. S. de; SILVA, T. L.; ALOISE, D. J. Otimização por nuvem de partículas: diferença entre aplicações a problemas contínuos e discretos. *XXXVI SBPO. São João Del-Rei–MG*, 2004.

PARPINELLI, R. S.; LOPES, H. S. New inspirations in swarm intelligence: a survey. *International Journal of Bio-Inspired Computation*, Inderscience Publishers Ltd, v. 3, n. 1, p. 1–16, 2011.

PARROTT, D.; LI, X. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *Evolutionary Computation, IEEE Transactions on*, IEEE, v. 10, n. 4, p. 440–458, 2006.

PASSINO, K. M. Biomimicry of bacterial foraging for distributed optimization and control. *Control Systems, IEEE*, IEEE, v. 22, n. 3, p. 52–67, 2002.

PÉTROWSKI, A. A clearing procedure as a niching method for genetic algorithms. In: IEEE. *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. 1996. p. 798–803.

PRICE, K.; STORN, R. M.; LAMPINEN, J. A. *Differential evolution: a practical approach to global optimization*. : Springer Science & Business Media, 2006.

RAND, W.; RIOLO, R. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: A case study using the shaky ladder hyperplane-defined functions. In: ACM. *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*. 2005. p. 32–38.

ROZENBERG, G.; BCK, T.; KOK, J. N. *Handbook of natural computing*. : Springer Publishing Company, Incorporated, 2011.

THOMSEN, R. Multimodal optimization using crowding-based differential evolution. In: IEEE. *Evolutionary Computation, 2004. CEC2004. Congress on*. 2004. v. 2, p. 1382–1389.

YANG, S.; LI, C. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *Evolutionary Computation, IEEE Transactions on*, IEEE, v. 14, n. 6, p. 959–974, 2010.

YANG, X.-S. Firefly algorithm. *Nature-inspired metaheuristic algorithms*, v. 20, p. 79–90, 2008.

YANG, X.-S.; HOSSEIN GANDOMI, A. Bat algorithm: a novel approach for global engineering optimization. *Engineering Computations*, Emerald Group Publishing Limited, v. 29, n. 5, p. 464–483, 2012.