UNIVERSIDADE FEDERAL DE ITAJUBÁ PROGRAMA
DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Análise e Desenvolvimento de Metodologias de Otimização Aplicadas em Sistemas Elétricos de Potência

Daniele de Alcântara Barbosa

Itajubá, outubro de 2009

Análise e Desenvolvimento de Metodologias de Otimização Aplicadas em Sistemas Elétricos de Potência

Por

Daniele de Alcântara Barbosa

M.Sc. (Automação e Sistemas Elétricos Industriais, UNIFEI) 2005

Tese submetida
para a Obtenção do Título
Doutor em Engenharia Elétrica

na área

Automação e Sistemas Elétricos Industriais

pelo

Programa de Pós-Graduação em Engenharia Elétrica
da
Universidade Federal de Itajubá, MG

Orientador(es):
Professor Leonardo de Mello Honório
Professor Antônio Carlos Zambroni de Souza

Itajubá 2009

Ficha catalográfica elaborada pela Biblioteca Mauá –
Bibliotecária Jacqueline Balducci - CRB_6/1698

B238m

Barbosa, Daniele de Alcântara.
Análise e Desenvolvimento de Metodologias de Otimização Aplicadas em Sistemas Elétricos / Daniele de Alcântara Barbosa. -- Itajubá, (MG) : [s.n.], 2009.
112 p. : il.

Orientador: Prof. Dr. Leonardo de Mello Honório.
Co-orientador: Prof. Dr. Antônio Carlos Zambroni de Souza.
Tese (Doutorado) – Universidade Federal de Itajubá.

1. Programação orientada a aspectos. 2. Sistema imunológico artificial . 3. Otimização. 4. Fluxo de Potência. I. Honório, Leonardo de Mello, orient. II. Souza, Antônio Carlos Zambroni de, co-orient. III. Universidade Federal de Itajubá. IV. Título.

Aprovado em

COMISSÃO DE AVALIAÇÃO

_____

_____

_____

_____

Universidade Federal de Itajubá

Itajubá 2009

Modelagem e Desenvolvimento de Métodos de Otimização em Sistemas Elétricos

Daniele de Alcântara Barbosa

Resumo

Este trabalho aborda o desenvolvimento de softwares para Fluxo Ótimo de Potência de duas formas: A primeira é através da modelagem e desenvolvimento de um software computacional mais flexível, onde mudanças em variáveis e funções objetivo sejam facilmente alteradas ou incorporadas. Isto é realizado através de um novo paradigma de modelagem e desenvolvimento de software denominado Orientação a Aspectos (Aspect-Oriented Programming AOP). A segunda é através do desenvolvimento de um algoritmo de otimização eficaz que já incorpore e minimize algumas das dificuldades existentes em outros métodos. Ou seja, uma abordagem foca a análise de software que pode ser aplicada a qualquer método de otimização e outra se concentra no desenvolvimento de um método de otimização específico que integre diversas características de outros sistemas.

Da mesma forma, este trabalho apresenta duas linhas gerais, uma para cada abordagem previamente descrita, onde a integração destas produz um paradigma eficaz para a resolução de fluxo de potência ótimo.

Abstract

This work presents two different approaches for optimal power flow software development: the first one deals with a more flexible software development paradigm based on Aspect-Oriented Programming (AOP) where the final code may be easily changed. The second approach presents an optimization algorithm that deals with some difficulties present in other methods. In that way, one approach deals with the software development that may use any optimization algorithm and the other presents a specific optimization methodology base on intelligent techniques.

Therefore this presents two different approaches that provide very powerful techniques for optimal power flow development.

.

Dedico a minha filha Bruna.

Índice

Lista de Figuras

Lista de Tabelas

Agradecimentos

Ao meu Orientador, Leonardo de Mello Honório, pela orientação na realização deste trabalho. Muito obrigada pela paciência, sabedoria e pelo incentivo constante.

Ao meu co-orientador Antônio Carlos Zambroni de Souza, pelos conhecimentos passados durante o desenvolvimento.

Ao Professor Armando Martins Leite da Silva, pela grande contribuição.

Aos colegas e professores que fazem parte do CRTI.

A todos vocês, meus sinceros agradecimentos.

1 Introdução

Normalmente, fluxo de potência ótimo (FPO) se apresenta como um problema não-linear, não convexo, de grande escala, podendo ter variáveis contínuas, discretas e, uma grande variedade de objetivos. Esta variedade de problemas faz com que o desenvolvimento destes sistemas seja difícil de implementar, além de específicos para determinados casos.

Existem duas formas principais de abordar estes problemas. A primeira é através da modelagem e desenvolvimento de um software computacional mais flexível, onde mudanças em variáveis e funções objetivo sejam facilmente alteradas ou incorporadas. A segunda é por meio do desenvolvimento de um algoritmo de otimização eficaz, que já incorpore e minimize algumas das dificuldades descritas anteriormente. Ou seja, uma abordagem foca a análise de software que pode ser aplicada a qualquer método de otimização e outra se concentra no desenvolvimento de um método de otimização específico que integre diversas características de outros sistemas.

Da mesma forma, este trabalho apresenta duas linhas gerais, uma para cada abordagem previamente descrita, e a integração destas produz um software eficaz para a resolução de fluxo de potência ótimo.

1.1 Modelagem e Desenvolvimento de Softwares em Sistemas de Potência

Um sistema de FPO [1] pode ser desenvolvido utilizando diversas tecnologias e paradigmas de programação. Porém, existe um contrapeso entre performance computacional e

custo de desenvolvimento de software; otimizações computacionais normalmente aumentam a complexidade das estruturas de programação e algoritmos gerando programas de alta complexidade, de difícil compreensão e manutenção.

Para solucionar parcialmente estes problemas, a literatura apresenta algumas modelagens de desenvolvimento de softwares baseados em orientação a objetos (POO). Por exemplo, em [2] POO é utilizado para modelar sistemas de potência com operações de matrizes esparsas, em [3] é utilizado em simulações de sistemas dinâmicos, em [4] é utilizado para restauração, em [5] POO é utilizado em conjunto com uma linguagem de modelagem unificada, Unified Modeling Language (UML), para uma descrição completa em termos de engenharia de software de um problema de despacho ótimo.

Para uma modelagem eficiente de POO é necessário dividir o problema em classes. Cada uma com métodos (funções) e propriedades (dados) bem definidos de uma determinada parte do problema. A junção destes módulos para gerar o sistema final é realizada através de mecanismos tradicionais como associação, composição, herança, polimorfismo, etc. Entretanto, estes mecanismos geram um acoplamento muito forte e específico entre as diversas partes do problema fundindo partes não muito relacionadas como, por exemplo, o problema de fluxo de potência com o problema de otimização. Este acoplamento é o grande responsável pelos problemas de escalabilidade (capacidade de um sistema crescer mantendo a complexidade sob controle) e manutenção. Uma analogia é o efeito dominó, onde mudanças em uma parte do código geram a necessidade de mudanças em várias partes do sistema.

Quebrar este acoplamento entre as classes tornaria o sistema muito mais flexível, com índices maiores de reutilização, já que mudanças em uma parte do código não afetariam as demais. Entretanto, mesmo utilizando as melhores práticas de engenharia de software, não é possível quebrar as complexas dependências de um FPO levando a necessidade de uma nova metodologia de modelagem e desenvolvimento. Isto leva à utilização de uma metodologia recém estruturada e inédita em sistemas elétricos de potência, que é a Programação Orientada a Aspecto, Aspect-Oriented Programming (AOP) [6,7,8].

AOP é um paradigma de programação desenvolvido para lidar com situações como as descritas anteriormente. AOP modela as classes do sistema, identifica como se acoplam e implementa as classes e os acoplamentos de forma separada. Desta forma, cada domínio específico de conhecimento gera suas respectivas classes, que são componentes normais desenvolvidos através da tradicional POO. Entretanto, a forma que estes domínios se ligam também é modelada, e estes novos mecanismos de ligação são denominados de aspectos funcionais, ou resumidamente aspectos. Esta modelagem pode ser realizada de forma estática ou dinâmica. Esta última é particularmente interessante e será adotada neste trabalho já não é necessário que se defina como as ligações devem ser implementadas antes do programa final ser gerado. Assim, é possível que um usuário, e.g. um operador do sistema elétrico, defina qual a função de otimização, o método e as variáveis a serem utilizadas pelo programa, durantes a execução do programa. Com os componentes tradicionais já desenvolvidos e com os aspectos fornecidos por um usuário, um sistema computacional desenvolvido, chamado de "Weaver" é responsável por compor e gerar as ligações finais. Como todo o sistema já estava previamente implementado, o AOP não interfere no desempenho computacional do sistema e proporciona uma flexibilidade muito grande.

Um dos objetivos deste trabalho é utilizar os benefícios do paradigma AOP para desenvolver uma solução de otimização de fluxo de potência altamente flexível. A idéia é quebrar o sistema de potência e os algoritmos de otimização em diversos componentes e aspectos, e evitar ao máximo qualquer tipo de dependência de relacionamento.

Através desta metodologia, o sistema AOP será capaz de exibir todas as informações úteis sobre variáveis, funções e métodos de otimização presentes nos aspectos, permitindo ao usuário ajustar e executar uma configuração desejada do FPO.

Este paradigma soluciona três dos maiores problemas relacionados com o desenvolvimento de softwares de FPO. Primeiro, restrições e variáveis de controle podem ser adicionadas ou removidas mais facilmente. Segundo, novas funções objetivo podem ser adicionadas automaticamente ao problema sem alterar praticamente nada os sistemas

computacionais já desenvolvidos. Finalmente, AOP quebra o contrapeso entre esforço de desenvolvimento e desempenho computacional. Uma vez que as classes e os aspectos são montados em conjunto de acordo com a necessidade, cada sistema gerado já está com o código otimizado para a melhor performance possível.

1.2 Algoritmos e Métodos de Otimização em Fluxo de Potência

Como bem documentado na literatura, as metodologias de resolução de problemas de FPO podem ser divididas em dois grandes grupos; numéricos e inteligentes. Considerando as metodologias numéricas a referência [9] traz uma comparação entre três métodos de pontos interiores; primal-dual (PD), preditor-corretor (PC) e múltipla correção de centralização (MCC). Os resultados mostram que todos os métodos tiveram bom desempenho, embora fosse necessário um ajuste criterioso de parâmetros para a resposta. Foram relatados também algumas dificuldades para problemas mistos (discreto e contínuo). Por exemplo, a referência [10] mostra que o tratamento de variáveis discretas, como contínuas, para o arredondamento no final do processo gera respostas com custos altos comparados a outras soluções. Para resolver este problema a mesma referência introduz uma melhoria no tratamento destas variáveis através de funções de penalidade. Porém, estes modelos tradicionais podem sofrer com escalabilidade mostrando-se inapropriados para sistemas de grande porte.

Já os métodos baseados em inteligência são alternativas interessantes para os problemas apresentados [11,12,13,14]. Várias vantagens podem ser associadas a estes algoritmos: a complexidade do software é baixa, são capazes de integrar facilmente variáveis contínuas e discretas, podem ter mais de uma solução como saída e apresentam um desempenho computacional razoável, especialmente se considerar que são métodos facilmente paralelizados e podem ser executados em clusters de computadores. Um dos problemas com a metodologia está no tratamento das restrições.

Um método muito comum apresentado na literatura [15,16] é o da função de penalidade (FP). Esta técnica converte o sistema primal restrito em um sistema irrestrito através da adição de uma função de penalidade. Embora seja a forma mais fácil de tratar as restrições, não é garantido encontrar um ponto estacionário ao final do processo de otimização. Para isto, as referências [17,18] apresentam métodos baseado no Lagrangeano. Estes, trabalham com os problemas primal e dual de forma separada gerando um problema Max-Min que pode ser resolvido através de uma abordagem multi-objetiva ou co-evolucionária. Uma vantagem é que estes métodos podem ser facilmente utilizados por qualquer algoritmo de otimização baseado em população. Porém, a grande maioria tem como característica oferecer apenas uma solução final.

Considerando a complexidade do setor elétrico, ter um algoritmo capaz de prover mais de uma solução é fundamental. Os principais algoritmos com esta funcionalidade são os baseados em computação evolucionária (evolutionary computation - EC) [19,20], e sistemas imunológicos artificiais (Artificial Immune Systems – AIS [11,21,22,23]

A metodologia AIS tem como princípio o sistema imunológico natural [21]. As características de um sistema imunológico podem ser utilizadas para aprendizado e otimização. Para este último, três tópicos são interessantes: proliferação, mutação e seleção. Enquanto a proliferação é a capacidade de gerar novos indivíduos fazendo o processo dinâmico de otimização, mutação é a habilidade de pesquisar dentro de um espaço de solução por pontos sub-ótimos, finalmente a seleção é responsável por eliminar células com baixa afinidade. Estas características fazem do AIS uma poderosa ferramenta, possibilitando a pesquisa para vários ótimos locais.

Existem diferentes metodologias de AIS para otimização. A Referência [21] mostra uma abordagem interessante que utiliza uma estratégia encontrada em alguns algoritmos evolutivos, clusters, no qual indivíduos são direcionados a pontos mais promissores no espaço de solução. Embora estes algoritmos tenham bons resultados, o número de indivíduos usados no processo de simulação é elevado, inadequado para problemas de FPO. Como, um dos objetivos deste

trabalho é demonstrar o conceito do AIS em problemas de FPO, algumas modificações no algoritmo proposto foram realizadas [11], como exemplo, a adição de informações mais relevantes aos indivíduos.

Em sistemas elétricos de potência, um indivíduo pode ser relacionado a uma condição operacional caracterizada pelas equações de fluxo de potência correspondentes, que descrevem seu comportamento elétrico. Entretanto, quando uma modificação em uma variável de controle é executada, é possível determinar o ponto operacional associado analisando a derivada das equações do fluxo de potência. Matematicamente, usando esta informação dada pelo vetor tangente (VT) é possível conduzir o processo de mutação, que no algoritmo original era completamente aleatório, e gerar melhores indivíduos, tornando o processo de otimização mais rápido e seguro.

Outra vantagem é que na metodologia AIS baseada em agrupamentos ("niching") todos os indivíduos influenciados pelo mesmo atrator (um ótimo local) irão convergir para um único ponto. Desta forma, com a correta identificação destes pontos é possível elimina durante o processo evolucionário, todos, exceto o melhor, de cada grupo sobre a influência de um dado atrator, reduzindo assim o esforço computacional.

1.3 Organização do Trabalho

Para demonstrar os conceitos e idéias o trabalho foi dividido nos seguintes capítulos: O capítulo 2 apresenta, além do problema de desenvolver sistemas de otimização para fluxo de potência ótimo, uma revisão bibliográfica sobre as metodologias a serem empregadas; tipos e métodos de otimização.

O capítulo 3 aborda sobre a engenharia de o software, as fases de desenvolvimento e introduz o desenvolvimento de um fluxo ótimo de potência (FPO) em diferentes paradigmas. Os capítulos 4 e 5 descrevem o paradigma orientado a aspectos e aplicação em sistemas de FPO.

O capítulo 6 traz as modificações e aprimoramentos realizados nas metodologias existentes de forma a melhorar o desenvolvimento de sistemas de otimização. Finalmente, O capitulo 7 mostra os resultados obtidos e as conclusões durante o desenvolvimento do trabalho .

2 Definição do Problema de Otimização em Sistemas Elétricos de Potência

2.1 Introdução

Não é objetivo deste trabalho aprofundar em sistemas elétricos de potência e comportamentos estáticos e dinâmicos. Para tanto, a Referência [24] é mais apropriada. Para melhor direcionar o leitor, os principais conceitos serão apresentados de forma reduzida.

Um sistema elétrico de potência é composto de vários equipamentos elétricos, como geradores, linhas de transmissão, transformadores, capacitores shunt, etc. O objetivo principal é gerar, transmitir e distribuir energia para consumidores através de um serviço de alta qualidade com custo mínimo, sujeitos às várias restrições físicas e operacionais. Visto que não é possível armazenar energia, a quantidade gerada em determinado momento, descontando as perdas de transmissão, deve ser igual a consumida. Em outras palavras, o balanço do fluxo de potência deve ser nulo. Para cumprir estas condições, operadores humanos precisam lidar com centenas, às vezes milhares de variáveis para controlar o sistema e direcionar o fluxo de energia para os consumidores. Além disto, durante o processo o sistema tem que permanecer fisicamente estável dos pontos de vista estático e dinâmico, o que é tecnicamente seguro e economicamente interessante para todos os agentes de mercado envolvidos.

As condições de balanço podem ser matematicamente descritas como:

$$\begin{gathered} y = M\,(x_{cl}) \\ g_i = (V, \theta, x_{cb}) - g_g + g_l = 0 \end{gathered} \tag{2.1}$$

onde: y é a matriz de admitância do sistema, x_{cl} são as variáveis de controle relacionado aos modelos dos elementos de transmissão (exemplo: linhas e transformadores); x_{cb} são as variáveis de controle relacionadas aos modelos das barras (exemplo: usinas de geração), g_i, g_g e g_l são vetores de injeção de potência, potência de geração e cargas para cada barra respectivamente. Se alguma mudança acontecer nas variáveis de controle, o sistema é direcionado a um ponto de operação diferente alterando valores de várias medidas, como tensão V e ângulo θ (nomeados como variáveis de estado), fluxo de potência ativa/reativa, etc., os quais serão difundidos ao longo da rede de potência. Encontrar o melhor ponto é um desafio. Este problema é nomeado sistema de fluxo ótimo de potência ou simplesmente FPO.

Um FPO é um problema de otimização não-linear, não-convexo que calcula um conjunto de variáveis ótimas de estado e controle da rede, a partir dos dados de carga e parâmetros do sistema. Pode ser representado matematicamente por:

$$\begin{gathered} Minimizar f(x), x = \{x_c, x_s\} \\ sujeito.a\ \ g(x) = 0 \\ H_{min} \leq g(x) \leq H_{max} \\ x_{min} \leq x \leq x_{max} \end{gathered} \tag{2.2}$$

onde: x_s é um vetor de variáveis de estado (tensão e ângulo nas barras de carga, etc.); x_c é um vetor de variáveis de controle (tensão e geração nas barras, potência de geração, "shunt" capacitor, transformadores taps, carga nas barras, etc.); f é uma função escalar representando a função objetivo a ser otimizada (i.e. despacho econômico, redução de perda na transmissão, carregamento, redução de carga); g representa o balanço das equações de fluxo de potência ativa

e reativa; e *h* é um vetor de restrições associado aos valores de funções como fluxo das linhas de transmissão, geração reativa, etc.

Existem vários problemas relacionados a solução de 2.2, como:

1. Apesar da existência de um ponto operacional, diversas variáveis podem ter seus limites violados. Em caso de contingência talvez seja impossível atender todos os limites, desta forma o método aplicado deve ser capaz de facilmente alterar sua formulação para inserir e remover inequações, ou seja, restrições. Este problema aumenta se um método numérico, como a técnica primal-dual, for utilizada [25]. Se, por exemplo: tentar invalidar uma dada restrição, aumentando seus limites.

2. Ações de controle indicadas por alguns métodos podem direcionar o ponto corrente a uma região completamente diferente, gerando problemas de estabilidade se tais ações forem tomadas de uma vez. Em uma aplicação real, onde vários controles precisam ser implementados para alcançar outro ponto operacional, as ações devem ser empregadas cuidadosamente. Maiores alterações podem conduzir a pontos de operação instáveis, podendo levar o sistema ao colapso. Conseqüentemente o método de otimização deve dar além do resultado final, a seqüência de como as ações devem ser tomadas.

3. Em cenários com altas cargas ou quando algum incidente ocorre, o espaço de solução é reduzido severamente, levando a uma situação onde os métodos baseados em população, dos quais dependem de ações aleatórias para gerar indivíduos válidos, não sejam mais eficazes.

4. A referência [10] mostra que o tratamento de variáveis discretas, como contínuas, até que elas estejam perto da solução ótima para então aproximá-las para o valor

correspondente mais próximo, pode fornecer resultados com um custo significantemente maior do que o apresentado pela solução ótima. Também sugere que técnicas que trabalham com programação inteira-mista podem sofrer com escalabilidade, mostrando inadequadas para problemas de grande escala.

Para projetar um FPO deve-se avaliar a capacidade de lidar com todos estes problemas. O sistema precisa ser operado de forma eficaz além de alcançar um bom ponto de operação. As variáveis de controle, função objetivo e restrições precisam ser testadas e o problema de escalabilidade avaliado. A Figura 2.1 ilustra este entrelaçamento de opções. Para o tratamento, diversas condições lógicas devem ser consideradas, atendendo a cada variação.

Considerando somente três métodos de otimização, uma grande quantidade de formulações pode ser produzida. No mínimo 192 possibilidades se somente um tipo de controle for utilizado, ou mais de 967.000 possibilidades (3x8x8!) se consider a combinação de mais controles. Para uma função objetivo específica, há mais de 80.000 possibilidades de soluções. O tratamento desta quantidade de opções requer que o código trabalhe com diversas condições lógicas.

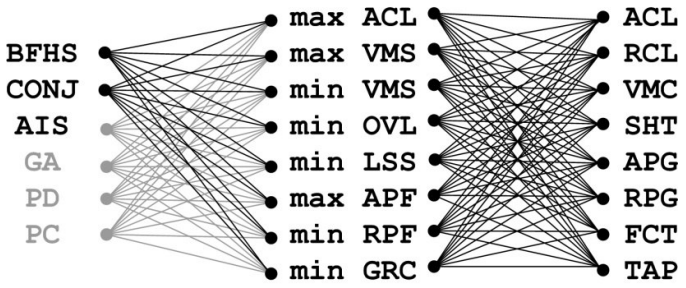

Figura 2.1- Comparação entre POO e AOP

Onde,

BFHS Método de Otimização BFHS (Best First Heuristic Search)

CONJ Método de Otimização CONJ (Gradiente Conjugado)

AIS Método de Otimização AIS (Artificial Intelligence System)

GA Método de Otimização GA (Genetic Algorithm)

PD Método de Otimização Primal-Dual

PC Método de Otimização Predictor-Corrector,

MAX Maximizar

MIN Minimizar

ACL Crescimento de Carga Ativa (Loadability) ou Corte de carga (Load Shedding)

APF Fluxo de Potência Ativo.

APG Controle de Geração de Potência Ativa.

FCT Controle de FACTS

GRC Custo de Geração (Economic Dispatch)

LSS Perdas

OVL Sobrecarga

RCL Controle de Carga Reativa

RPF Fluxo de Potência Reativo

RPG Controle de Geração de Potência Reativa.

SHT Shuntl

TAP TAP

VMC Controle de Tensão

VMS Valor de Tensão.

Desta forma seria interessante desenvolver um sistema computacional capaz de trocar facilmente a função objetivo, restrições, variáveis de controle e o método de otimização.

2.2 Otimização de Sistemas

Otimização é o processo de buscar a melhor solução (ou solução ótima) em determinado problema. Matematicamente, refere-se a minimização ou maximização de uma dada função objetivo sobre um conjunto de variáveis de decisão que satisfaçam a restrições funcionais.

A função objetivo é o que se deseja otimizar (minimizar ou maximar), como exemplo: minimizar perdas, maximizar lucro. Ela permite fazer comparações dentre as possíveis escolhas das variáveis e então determinar a "melhor". Variáveis de decisão são os parâmetros cujos valores determinam um resultado para o problema. Finalmente as restrições são definidas como um conjunto de funções que determinam o espaço factível das soluções.

Problemas com ausência de restrições são denominados irrestritos, enquanto que os demais são referenciados como problemas de otimização restrita. É possível encontrar problemas sem função objetivo, onde o intuito é apenas encontrar decisões que sejam viáveis. Já outros possuem múltiplas funções. Estes, geralmente são reduzidos a apenas um objetivo, combinando-se os múltiplos problemas em apenas um, ou escolhendo um objetivo e introduzindo restrições.

Se em um problema as variáveis são restritas a valores inteiros ou a um conjunto de possibilidades discretas, o problema é de otimização discreta. Caso não se restrinja a estes valores, podendo assumir valores reais, o problema é conhecido como otimização contínua.

Atualmente, existem inúmeros estudos em otimização, sempre com o objetivo de encontrar os melhores métodos para representar de maneira eficiente o problema analisado. Neste contexto, o conhecimento da forma que um problema de otimização se apresenta e como este deve ser solucionado é um fator relevante de sucesso. A seguir, os principais modelos de problemas de otimização são descritos com base nos conceitos das referências [26,27,28,29].

2.3 Definição Matemática do Problema de Otimização

Dada uma função $f(x): \mathbb{R}^n \to \mathbb{R}$ e um conjunto $S \subset \mathbb{R}^n$, deve-se encontrar um $x^* \in \mathbb{R}^n$ que resolva 2.3.

$$\begin{matrix} \min_x f(x) \\ s.a\ x \in S \end{matrix} \tag{2.3}$$

onde, f é considerada a função objetivo e S como uma região factível.

Se S é vazio, o problema é chamado infactível. Se é possível encontrar uma seqüência $x^k \in S$ onde e $f(x^k) \to -\infty$ e $k \to +\infty$, então o problema é ilimitado. Se o problema é não factível nem ilimitado, em muitos casos é possível encontrar uma solução $x^* \in S$ que satisfaça a equação:

$$f(x^*) \leq f(x), \forall x \epsilon S \tag{2.4}$$

onde x^* é um minizador global do problema 2.3. Já se

$$f(x^*) < f(x), \forall x \epsilon S, x \neq x^*, \tag{2.5}$$

Dizemos que se trata de um minimizador global restrito.

Em muitos casos, um conjunto factível S é descrito usando restrições funcionais de igualdade e desigualdade. Por exemplo, S pode ser dado com:

$$S: = \{x: g_i(x) = 0, i \in \varepsilon \text{ e } g_i(x) \geq 0, i \in \mathcal{I}\}, \tag{2.6}$$

onde ϵ e $\mathcal{I}$ são índices dos conjuntos das restrições de igualdade e desigualdade. A equação 2.7 mostra a forma genérica do problema de otimização.

$$\begin{aligned} \min_x \quad & f(x) \\ & g(x) = 0, \\ & h(x) \geq 0, \quad . \end{aligned} \tag{2.7}$$

Muitos fatores que afetam os problemas de otimização podem ser solucionados eficientemente. O número n de variáveis de decisão e o número total de restrições $|\varepsilon|+|\tau|$, são geralmente bons indicadores para analisar a dificuldade da solução de determinados problemas. Outros fatores estão relacionados às propriedades das funções *f* e *g*.

Sabe-se que problemas com a função objetivo linear e restrições lineares são mais fáceis de resolver, outros já não são triviais. Por esta razão, pesquisadores têm desenvolvido diferentes algoritmos com características cada vez mais específicas para cada caso.

A seguir uma breve descrição dos principais tipos de otimização.

2.3.1 Otimização Linear

Um dos problemas mais comuns e fáceis de otimização é a linear, ou programação linear (PL). Em um problema de PL a função objetivo e as restrições, sejam elas de igualdades ou desigualdades, são lineares. Caso as funções *f* e g_i apresentadas pela equação 2.7 sejam todas lineares, o problema é de PL.

A forma padrão de qualquer é dada pela equação 2.8.

$$\begin{aligned} &\min_x c^T x \\ &Ax = b \\ &x \geq 0, \end{aligned} \tag{2.8}$$

Onde $A \in \mathbb{R}^{mx,n}, b \in \mathbb{R}^{m}, c \in \mathbb{R}^{n}$ são valores constantes conhecidos, e $x^{*} \in \mathbb{R}^{n}$ é o vetor de variáveis a ser determinado. Existem diferentes métodos para programação linear, os melhores e mais conhecidos são os métodos de pontos interiores e o simplex.

2.3.2 Otimização Não-Linear

Na programação não linear (PNL), os problemas têm ou a função objetivo ou algumas das restrições não lineares. Caso *f*, ou alguma das funções de *g* apresentadas pela equação 2.7 for não linear, o problema é da PNL.

Podem ser solucionados utilizando técnicas de pesquisa pelo gradiente, bem como métodos baseados em Newton, métodos de pontos interiores, métodos programação quadrática, dentre outros. Podem ser irrestritas ou restritas.

Otimização sem restrições (irrestrita): Classe de problemas que possuem função objetivo não linear e não possuem restrições. Constitui um bloco fundamental no desenvolvimento de software. Considere o problema de minimizar uma função f(x), $x \in \mathbb{R}^{n}$. A equação 2.9 define o problema como:

$$\begin{matrix} \min \quad f(x) \\ x \in \mathbb{R}^{n}, \end{matrix} \tag{2.9}$$

Otimização com restrições (restrita): No problema não-linear de otimização com restrições, a função objetivo e as funções de restrições são função não lineares gerais de *x*. O problema geral é dado pela equação 2.10.

$$\begin{matrix} \min_{x} f(x) \\ s.a\ g(x) \leq 0 \\ x \in \mathbb{R}^{n} \end{matrix} \tag{2.10}$$

2.4 Algoritmos de Otimização

Como descrito anteriormente, métodos de otimização são ferramentas de análise essencial quando se é necessário tratar com problemas de pesquisa em espaço de soluções complexos, dada uma ou mais funções objetivo. Estes problemas são encontrados nas mais diferentes áreas, com diferentes características como: processos químicos, pesquisa operacional, sistemas elétricos, etc. Para o desenvolvimento de fluxo de potência, a literatura adota basicamente dois conjuntos de soluções: métodos numéricos matemáticos e implementações que utilizam técnicas de inteligência.

Cada método tem suas vantagens e são indicados conforme a necessidade do problema. A seguir serão apresentados os principais conceitos.

2.4.1 Numéricos

Os métodos numéricos matemáticos são práticas largamente utilizadas em diversos problemas. Diferente dos inteligentes, nestes existe a necessidade em ter o modelo matemático do sistema implementado.

Uma solução confiável e com alto desempenho para estes métodos envolve um conjunto baseado em pontos interiores (PI). Uma das mais poderosas técnicas presentes na literatura é o método primal-dual (PDPI) de pontos interiores.

O PDPI basicamente trata de um problema de otimização onde uma função lagrangeana é montada, e então utiliza o método de Newton Raphson para resolvê-la. A idéia intuitiva do método dos pontos interiores consiste em buscar a solução ótima reduzindo a função objetivo, no caso de minimização, mantendo a busca no interior da região delimitada pelas restrições [29]. Existem grandes diferenças entre usar métodos numéricos e inteligentes, as mais consideradas são o custo de desenvolvimento e o tempo de convergência.

2.4.2 Inteligentes

A inteligência Artificial (IA) é uma área do conhecimento que utiliza o comportamento humano como inspiração para desenvolver sistemas inteligentes, com ênfase na representação do conhecimento e em métodos de inferência. A principal razão de utilizar estes métodos no desenvolvimento está na possibilidade de resolver problemas complexos, que envolvam diversas variáveis.

Os métodos inteligentes possibilitam a determinação de soluções factíveis para uma enorme gama de problemas encontrados. A principal dificuldade a ser vencida pelos métodos numéricos reside na necessidade de uma busca exaustiva a ser realizada no espaço de solução dos problemas. Já os inteligentes tentam superar esta questão guiando a geração de soluções de modo a proporcionar uma solução de qualidade (avaliados seguindo funções específicas) em um tempo computacional limitado.

Dentre as vantagens destes métodos estão: a complexidade do software é simples, são capazes de trabalhar com variáveis inteiras e contínuas, apresentam bom desempenho computacional.

A seguir será apresenta uma breve descrição dos métodos utilizados neste trabalho.

- **Busca em Árvore**

Os métodos de busca em árvore podem ser utilizados quando existem opções com valores conhecidos, tendo por objetivo encontrar a melhor seqüência para um determinado problema. O procedimento inicia com um nó raiz sendo o estado inicial, e a partir deste é gerada uma árvore com os estados possíveis. Existem diversas estratégias de busca para esta metodologia, como busca em largura, profundidade, A^*, etc. Duas técnicas, apresentadas em [1] serão mostradas a seguir.

- **"Best First Heuristic Search"**

BFHS é uma técnica de otimização que busca pela melhor solução analisando todos os nós promissores de acordo com alguma regra heurística. Esta procura expande o nó que é mais próximo ao objetivo, acreditando numa condução rápida ao objetivo. Todas as ações disponíveis são analisadas e somente uma é escolhida.

São necessárias as seguintes estruturas para implementação:

OPEN: nós que já foram gerados e avaliados por uma função heurística, mas ainda não foram examinados. Pode ser vista como uma fila com prioridades.

*CLOSED: nós que já foram examinados (caso de grafos).

A função heurística é convenientemente definida como: $f' = g(n) + h(n)$.

onde: f' é a aproximação de uma função f que representa a avaliação real do nó, g é uma estimativa do custo do caminho ótimo a partir do nó inicial até o nó atual, h é uma estimativa do custo do nó atual até o nó objetivo.

Se mais de um caminho gerar o nó, o algoritmo irá armazenar o melhor deles.

Para o presente trabalho, o algoritmo BFHS é aplicado em problemas de FPO e os conceitos descritos a seguir são necessários:

Um ponto operativo factível (POF): considerando um caso que opera sob alta tensão, POF é definido como um ponto operativo estável que considera somente os limites relativos ao controle de amplitude

Função heurística(FH) : a FH estima o custo do caminho para mover o POF corrente para outro mais perto da função objetivo. Devido o sistema ser não-linear, é possível que a direção mude continuamente. FH é calculado através do vetor tangente (VT) O VT dá a sensibilidade da variável de estado com relação a alterações na variável de controle. A principal razão de calcular a sensibilidade é de comparar todas as ações disponíveis e descobrir qual controle é mais provável para influenciar a função objetivo.

Ações de controle disponíveis (open nodes): analisando todas as ações possíveis a cada passo exigiria muito tempo, especialmente se considerar que grande partes destes controles seriam ineficazes. Assim, a cada passo, o FH indica a ação mais eficaz para alcançar o objetivo, as ineficazes são descartadas.

- **Gradiente Conjugado**

Quando ações mais complexas precisam ser executadas o método BFHS, exibido no tópico acima não é mais apropriado.

O método do gradiente conjugado, uma das técnicas de otimização irrestrita da programação matemática é similar ao BFHS. Deve ter um ponto de operação factível, um conjunto de ações de controle disponíveis e uma função heurística que controla a sensibilidade a fim de permitir (ou não) uma respectiva ação. Porém, o CONJ dá um passo em toda ação disponível.

Pode-se dizer que o gradiente conjugado é o método das direções conjugadas que consiste na seleção de sucessivos vetores direção como uma versão conjugada dos sucessivos gradientes encontrados ao longo do processo de solução. É aplicado em diversos tipos de problemas, normalmente na solução de problemas de controle ótimo com convergência razoavelmente rápida e economia de memória.

- **Sistema Imunológico Artificial**

O sistema imunológico artificial (AIS) é um método baseado no sistema imunológico natural (SIN). Tem como objetivo capturar alguns aspectos do SIN e implementá-los utilizando técnicas e padrões de projetos computacionais.

Como todo método baseado em inteligência, o SIA é uma metodologia de pesquisa que usa heurística para explorar áreas de interesse no espaço de solução. Dentre as vantagens do método está a capacidade de executar simultaneamente buscas locais e globais.

Neste trabalho, será utilizado um desenvolvimento inédito de um algoritmo Imunológico artificial, capaz de acrescentar várias funcionalidades e vantagens aos algoritmos apresentados anteriormente. Serão agregados os seguintes elementos; a) um gradiente numérico avaliará de forma mais eficaz a evolução da população, b) será feita a identificação de indivíduos redundantes (que se dirigem para o mesmo ponto estacionário) visando reduzir a população, e c) uma nova forma de lidar com as restrições irá garantir a convergência do sistema para um ponto estacionário. Todos estes novos pontos serão detalhados no capítulo 6.

3 Análise e Desenvolvimento de Sistemas

Dentre as fases de desenvolvimento de um software, serão mencionados a análise e desenvolvimento para o sistema de fluxo de potência ótimo. Os principais conceitos, dificuldades e as etapas do desenvolvimento serão apresentadas. O objetivo deste capítulo é deixar clara a importância da orientação a aspecto dentro do contexto estudado.

3.1 Definição de Software

Pode-se definir o software, numa forma clássica, como sendo: "um conjunto de instruções que, quando executadas, produzem a função e o desempenho desejados, estruturas de dados que permitam que as informações relativas ao problema a resolver sejam manipuladas adequadamente e a documentação necessária para um melhor entendimento da sua operação e uso".

Entretanto, no contexto da Engenharia de Software, o software deve ser visto como um produto a ser "vendido". É importante dar esta ênfase, diferenciando os "programas" que são concebidos num contexto mais restrito, onde o usuário ou "cliente" é o próprio autor. No caso destes programas, a documentação associada é pequena ou (na maior parte das vezes) inexistente e a preocupação com a existência de erros de execução não é um fator maior, já que o principal usuário é o próprio autor do programa, e este não terá dificuldades, em princípio, na detecção e correção de um eventual "bug". Além do aspecto da correção, outras boas

características não são também objeto de preocupação como a portabilidade, a flexibilidade e a possibilidade de reutilização.

Um produto de software por outro lado, é sistematicamente destinado ao uso de outras pessoas, e não os seus programadores. Os eventuais usuários podem, ainda, ter formações e experiências diferentes, o que significa que uma grande preocupação no que diz respeito ao desenvolvimento do produto deve ser a sua interface, reforçada com uma documentação rica em informações para que todos os recursos oferecidos possam ser explorados de forma eficiente. Ainda, os produtos de software devem passar normalmente por uma exaustiva bateria de testes, dado que os usuários não estarão interessados (e nem terão capacidade) de detectar e corrigir os eventuais erros de execução.

Resumindo, um programa desenvolvido para resolver um dado problema e um produto de software destinado à resolução do mesmo problema são duas coisas totalmente diferentes. É óbvio que o esforço e o conseqüente custo associado ao desenvolvimento de um produto serão muito superiores.

Em função destas características diferenciais, o processo de desenvolvimento de software pode desembocar em um conjunto de problemas, os quais terão influência direta na qualidade do produto.

Tudo isto porque, desde os primórdios da computação, o desenvolvimento dos programas (ou, a programação) era visto como uma forma de arte, sem utilização de metodologias formais e sem qualquer preocupação com a documentação, entre outros fatores importantes. A experiência do programador era adquirida através de tentativa e erro. A verdade é que esta tendência ainda se verifica. Com o crescimento dos custos de software (em relação aos de hardware) no custo total de um sistema computacional, o processo de desenvolvimento de software tornou-se um item de fundamental importância na produção de tais sistemas.

Uma vez que um software se resume a um conjunto de comandos e procedimentos para executar uma dada tarefa, é fundamental para a realização de um bom produto ter a noção exata das tarefas que o sistema deve realizar. Um mau dimensionamento das tarefas leva a um mau

desenvolvimento de código, que por sua vez leva a um mau produto final. Com a qualidade final baixa é bem provável que o sistema tenha que ser refeito e o custo disso é desastroso. A Figura 3.1 mostra o impacto do mau levantamento dos requisitos no custo final de um desenvolvimento. O simples conhecimento deste problema não significa que uma equipe vá conseguir evitá-lo, mas, sem dúvida, é um primeiro passo para a sua solução. Em primeiro lugar, é preciso estar ciente também de que não existe uma abordagem mágica que seja a melhor para a solução dos problemas, mas uma combinação de métodos que sejam abrangentes a todas as etapas do desenvolvimento de um software.

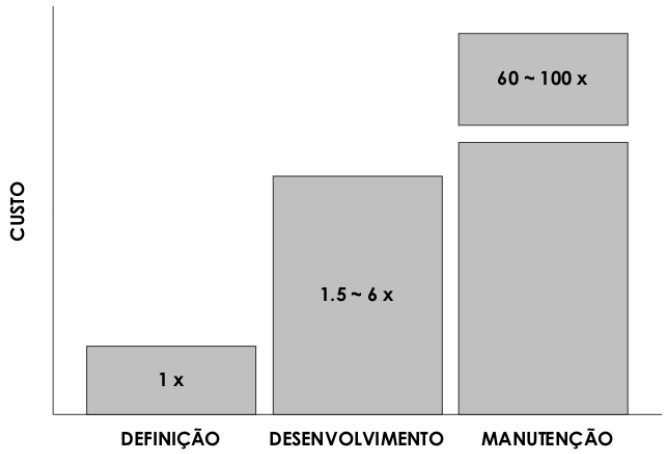

Figura 3.1- Influência das Alterações de Requisitos no Custo de um Sistema

As fases apresentadas na Figura 3.1, são detalhadas a seguir.

3.1.1 Fase de Definição

A fase de definição está associada à determinação do que vai ser feito. Nesta fase, o profissional encarregado do desenvolvimento do software deve identificar as informações que deverão ser manipuladas, as funções a serem processadas, qual o nível de desempenho desejado,

que interfaces devem ser oferecidas, as restrições do projeto e os critérios de validação. Isto terá de ser feito não importando o modelo de desenvolvimento adotado para o software e independente da técnica utilizada pra fazê-lo.

Esta fase é caracterizada pela realização de três etapas específicas:

- Análise (ou Definição) do Sistema, a qual vai permitir determinar o papel de cada elemento (hardware, software, equipamentos, pessoas) no sistema, cujo objetivo é determinar, como resultado principal, as funções atribuídas ao software;
- Planejamento do Projeto de Software, no qual, a partir da definição do escopo do software, será feita uma análise de riscos e a definição dos recursos, custos e a programação do processo de desenvolvimento;
- Análise de Requisitos, que vai permitir determinar o conjunto das funções a serem realizadas assim como as principais estruturas de informação a serem processadas.

3.1.2 Fase de Desenvolvimento

Nesta fase, será determinado como realizar as funções do software. Aspectos como a arquitetura do software, as estruturas de dados, os procedimentos a serem implementados, a forma como o projeto será transformado em linguagem de programação, a geração de código e os procedimentos de teste devem ser encaminhados.

Normalmente, esta fase é também organizada em três principais etapas:

- Projeto de Software, o qual traduz, num conjunto de representações gráficas, tabulares ou textuais, os requisitos do software definidos na fase anterior; estas representações (diversas técnicas de representação podem ser adotadas num mesmo projeto) permitirão definir, com um alto grau de abstração, aspectos do

software como a arquitetura, os dados, lógicas de comportamento (algoritmos) e características da interface;

- Codificação, onde as representações realizadas na etapa de projeto serão mapeadas numa ou em várias linguagens de programação, a qual será caracterizada por um conjunto de instruções executáveis no computador. Nesta etapa, considera-se também a geração de código de implementação, aquele obtido a partir do uso de ferramentas (compiladores, linkers, etc...) e que será executado pelo hardware do sistema;
- Testes de Software, onde o programa obtido será submetido a uma bateria de testes para verificar (e corrigir) defeitos relativos às funções, lógica de execução, interfaces, etc.

3.1.3 Fase de Manutenção

A fase de manutenção, que se inicia a partir da entrega do software, é caracterizada pela realização de alterações de naturezas as mais diversas, seja para corrigir erros residuais da fase anterior, para incluir novas funções exigidas pelo cliente, ou para adaptar o software a novas configurações de hardware.

Sendo assim, pode-se caracterizar esta fase pelas seguintes atividades:

- Correção ou Manutenção Corretiva, a qual consiste da atividade de correção de erros observados durante a operação do sistema;
- Adaptação ou Manutenção Adaptativa, a qual realiza alterações no software para que ele possa ser executado sobre um novo ambiente (CPU, arquitetura, novos dispositivos de hardware, novo sistema operacional, etc...);

- o Melhoramento Funcional ou Manutenção Perfectiva, onde são realizadas alterações para melhorar alguns aspectos do software, como por exemplo, o seu desempenho, a sua interface, a introdução de novas funções, etc...

A manutenção do software envolve, normalmente, etapas de análise do sistema existente (entendimento do código e dos documentos associados), teste das mudanças e teste das partes já existentes, o que a torna uma etapa complexa e de alto custo.

O tempo proporcional e total investido em cada uma das fases depende principalmente do paradigma de programação; programação estruturada, orientação a objetos e orientação a aspectos. Uma descrição de cada um destes paradigmas utilizando o problema de fluxo de potência será apresentada a seguir.

3.2 Desenvolvimento de um Sistema de FPO com a Programação Estruturada

Atualmente existem diversos compiladores, alguns utilizam a programação estruturada (PE), como por exemplo, o FORTRAN e o Matlab. Estes ambientes podem fornecer bibliotecas com funções específicas, apresentam alto nível de abstração, e a princípio, são fáceis de serem utilizados.

A Figura 3.2 mostra um diagrama simplificado do uso FPO para este paradigma.

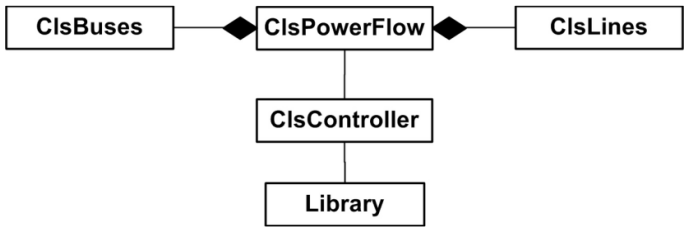

Figura 3.2- Projeto de um FPO Utilizando Bibliotecas Funcionais

Embora o diagrama pareça bastante simples, existe uma grande complexidade ocultada na classe *ClsPowerFlow* (Figura 3.2), responsável pelo tratamento de um FPO.

Para avaliar todos os cenários possíveis deveria ser implementado um código para cada situação, conforme o grafo ilustrado pela Figura 3.3. Considerando que exista pelo menos um bilhão de combinações possíveis seria impossível desenvolver um sistema com capacidade total. Esta conclusão está conforme os maiores softwares desenvolvidos no paradigma. Eles fornecem apenas algumas funções objetivo e somente controlam as principais variáveis. A Figura 3.3 mostra o grafo com todos os tratamentos lógicos necessários para avaliar uma dada função objetivo e acessar as variáveis de controle. Os nós na cor preta são regiões designadas para o tratamento de funções, e os nós em branco são para tratamento das variáveis de controle. Cada variável de controle gera 2 nós e 3 arestas. O número de nós e arestas gerados para cada função objetivo são aproximadamente de 2 a 3 vezes o número de ações de controle disponíveis.

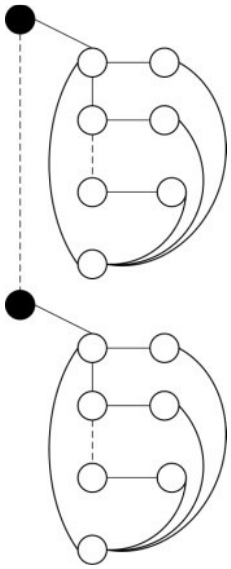

Figura 3.3- Grafo que Representa a Parte de Softwares de FPO para a PE

3.3 Desenvolvimento de um sistema de FPO usando a POO

Considere um sistema de FPO onde o usuário pode escolher o método de otimização, a função objetivo, variáveis de controle e limites. A Figura 3.4 ilustra de forma simplificada o modelo conceitual do sistema.

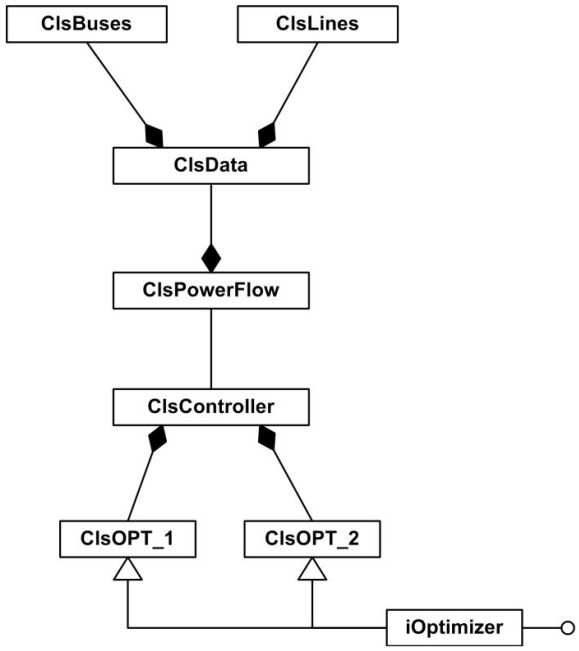

Figura 3.4- Diagrama Conceitual simplificado de POO-FPO

Existem vários caminhos para se projetar um FPO utilizando a POO. O projeto proposto é baseado nas melhores práticas de engenharia de software, isto é, ele utiliza um número razoável de herança e despacho dinâmico apenas nas maiores entidades do sistema, evitando uma explosão no número de classes.

O diagrama POO mostra o projeto centrado nas classes *ClsPowerFlow* e *ClsController*. A classe *ClsPowerFlow* contêm as informações matemáticas e físicas do sistema elétrico de potência. Existe um forte relacionamento desta classe com as variáveis, representadas por

ClsBuses e *ClsLines*. A classe *ClsPowerFlow*, junto com suas dependentes, tem a capacidade de solucionar um fluxo de potência tradicional, portanto é impossível sua separação.

Prosseguindo com a análise da outra parte do diagrama, as classes *ClsOPT*_1 e *ClsOPT*_2, responsáveis pela implementação dos métodos de otimização, apresentam um forte relacionamento com a interface *iOptimizer*. Esta interface é usada em todos os métodos de otimização. Desta forma é possível o emprego do despacho dinâmico (DD) assegurando um baixo grau de manutenção devido ao fato de que, se qualquer outra classe de otimização for adicionada no sistema, a única parte que deverá ser alterada é a que possui o código do tratamento lógico (*ClsController*), as outras classes e o resto do código permanecerão intactos.

A classe *ClsController* contém todas as variáveis relativas ao método de otimização: a função objetivo, as restrições e as variáveis de controle. Esta classe é responsável por coordenar todo o tratamento lógico que será utilizado pelo sistema. Este tratamento é dividido em três partes. O algoritmo 1 mostra a parte do *clsController* na qual o método de otimização e a função objetivo são instanciados, o algoritmo 2 mostra a parte que lida com a seleção das variáveis de controle e o algoritmo 3 mostra a parte que invoca o método de otimização desejado.

Algoritmo 1 *Parte 1 do código da classe controller*

```
public ClsController
…
itOptimizer optimizer;
DlgFunction dFdx;
switch (method) {
case "CON":
       optimizer = new ClsOOPCON();
       break;
case "BFHS":
       optimizer = new ClsOOPBFHS();
       break;… }
       …
switch (objectivefunction) {
case "f_PowerFlowAtualization":
       optimizer.OFunction =  new DlgFunction
                        (nOPFlow.f_FlowAtualization);
       break;
```

```
case "f_TransmissionFlow":
      … break;

 . . . }
```

Algoritmo 2 *Parte 2 do código da classe controller*

```
for (int i = 0; i < Length; i++){
switch (Ctrlvariable[i]){
case "C_XiFact":
      Ctrl[i]=ObPFlow.obLines.C_XiFact;
      break;
case "C_Tap":
      Ctrl[i]=ObPFlow.obLines.C_Tap;
      break;
case "S_ActivePowerFlow":
      Ctrl[i]=ObPFlow.obLines.S_ActivePowerFlow;
     break;
case "S_Loss":
      Ctrl[i]=ObPFlow.obLines.S_Loss;
      break;
. . . }
```

Algoritmo 3 *Parte 2 do código da classe controller*

```
switch (method) {
case "CON":
 optimizer.run(); break;
 …
case "PDIP":
if (ObPFlow.Check_For(f_dFdx_FlowAtualization))
{
      dFdx = DlgFunction(nOPFlow.f_dFdx_FlowAtualization);
      Optimizer.run(dFdx,PARAMETERS)
}
Else
{
      Return Error("unvalid dFdx");
}
break;… }
```

A análise do algoritmo 1 mostra as duas técnicas da POO que são bem conhecidas e úteis. (1) Os métodos de otimização são classes específicas que utilizam a interface *iOptimizer*. (2) a função objetivo é programada utilizando o método *delegates*, uma técnica similar ao uso de ponteiros em C, permitindo passar métodos sem necessidade de saber em tempo de compilação qual será invocado. O código do algoritmo 2 ajusta as variáveis de controle e estado de acordo com a especificação do usuário. O algoritmo 3 mostra uma das principais desvantagens de se utilizar a metodologia da POO quando é necessário uma lógica um pouco mais complexa; alguns métodos de otimização necessitam parâmetros diferentes que outros, por exemplo, se um método de ponto interior é escolhido, é necessário ter a primeira e segunda derivadas da função objetivo. Entretanto, se outro método for escolhido, as derivadas não são necessárias. Para trabalhar com estas possibilidades, o software de POO deve prover verificações no código como no algoritmo 2.

A Figura 3.5 mostra o grafo correspondente aos códigos presentes nos algoritmos 1 e 3 e a Figura 3.6 mostra o grafo do algoritmo 2. É claro que, analisando as figuras, a POO quebra o crescimento exponencial da complexidade, entretanto não evita o extensivo tratamento lógico para a solução. Consequentemente, se qualquer outro método de otimização, função objetivo ou variável de controle for adicionado ou removido, irá afetar o código das classes existentes.

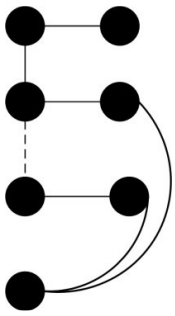

Figura 3.5- Grafo que Representa os Entrelaçamento Lógico dos Algoritmos 1 e 3

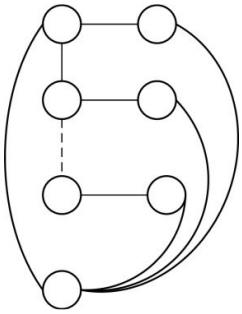

Figura 3.6- Grafo que Representa os Entrelaçamento Lógico do Algoritmo 2

3.4 Modelagem com Aspectos e Reflection

Como mostrado anteriormente, a programação estruturada é baseada na abstração de funções. No problema abordado, bibliotecas de funções são capazes de trabalhar de forma adequada com qualquer domínio do conhecimento, entretanto o tratamento lógico necessário para a utilização de vários domínios levaria a um crescimento exponencial do código, inviabilizando a utilização de diversas opções.

Com resultados muito melhores do que a PE, a modularização exercida na orientação a objetos é baseada na identificação e classificação de funções e variáveis. Esta característica, junto com ferramentas de composição mais avançadas como herança, polimorfismo e despacho dinâmico, aumentaram a abstração e proporcionaram um melhor tratamento lógico dos problemas. Entretanto, a modularidade gerada pelo POO tem um forte relacionamento com a forma que os principais conceitos envolvidos foram separados. Diferentes funcionalidades provenientes de diferentes domínios de conhecimento que possuem fortes conexões lógicas podem gerar um cruzamento de informações que se espalham por diversas classes. Este cenário é particularmente prejudicial quando o sistema deve ser evoluído constantemente. Isto é exatamente o que acontece em um software de sistemas de potência. Novos métodos de otimização, funções objetivos e restrições são constantemente adicionados ao problema já existente, exigindo uma manutenção periódica no software. Uma metodologia capaz de quebrar este tratamento lógico é a modelagem orientada a aspectos (AOP).

Um projeto baseado em AOP consiste em dois tipos de componentes; os baseados em POO e os componentes aspectuais. Para determinar qual parte do problema e que componente é necessário, em um primeiro momento é preciso modelar e decompor o sistema de acordo com o domínio do conhecimento. Cada domínio é estruturado de acordo com o paradigma POO junto com as ferramentas já mencionadas de abstração e composição. Os aspectos são responsáveis por descrever as características de alto nível do sistema integrado, ou seja, funcionalidades que podem ser alteradas, ou que ainda não foram consideradas, mas que possuem um entrelaçamento muito grande com os componentes já desenvolvidos.

O que irá diferenciar um aspecto de um componente regular é sua composição com o resto do sistema. Um componente regular pega um elemento do modelo e o subdivide usando relacionamentos internos e externos. Um aspecto pega um elemento do modelo e o subdivide usando apenas relações internas. As conexões entre os aspectos e o resto do sistema são definidas por um terceiro elemento, o *binder*. O *binder* usa a meta-informação dos componentes, como suas classes, seus métodos, etc, para poder compor o sistema. A composição é realizada através de um sistema a parte, denominado *Weaver*, que une as instruções de composição de forma a montar o sistema final.

Os capítulos 4 e 5 apresentam um detalhamento completo da programação orientada a aspectos, com os principais conceitos e o modelo adotado.

4 Programação Orientada a Aspectos

A Programação Orientada a Aspectos (AOP - Aspect-Oriented Programming) é um novo paradigma para facilitar o desenvolvimento e manutenção de softwares. É considerada uma das mais promissoras técnicas para melhorar a Orientação a Objetos (POO). Surgiu da necessidade de diminuir o acoplamento entre os componentes no desenvolvimento de sistemas com alta complexidade. Tem como principais colaboradores Gregor Kickzales e Cristina Lopes [30,1], também responsáveis pelo AspectJ [31,32], uma das primeiras implementações práticas de AOP nos anos 90.

Basicamente, o desenvolvimento de um sistema consiste em definir e implementar três níveis de soluções. Conceitual, que define a estrutura operacional do sistema onde, de acordo com a metodologia empregada, o paradigma escolhido do sistema é desenvolvido, Operacionais, que definem as funcionalidades do sistema, e não operacionais, que definem aspectos de tratamento de erro, memória, segurança, etc. Para este trabalho apenas os níveis Conceitual e Operacional serão definidos.

No paradigma tradicional de POO a idéia principal é identificar as características comuns e colocá-las dentro de classes individuais, cada uma contendo métodos e propriedades bem definidas como parte do problema. Mecanismos tradicionais como herança, agregação, composição são empregados. Porém, somente estes conceitos não asseguram um bom desenvolvimento de software, já que é impossível desacoplar métodos que atravessam várias funcionalidades do sistema. Embora seja possível subdividir o problema em classes apropriadas, dependendo da complexidade do sistema, o resultado é um complicado grafo de dependências.

Neste paradigma, o baixo acoplamento entre as classes também reduz a flexibilidade e dificulta a manutenção, já que a alteração de um código em uma classe, necessariamente não implica em mudanças nas demais. Todos estes problemas reduzem as vantagens da POO.

Para trabalhar com estas dificuldades surgiu a metodologia AOP. Ela complementa a POO e eliminam as dificuldades descritas acima. . O paradigma suporta novos mecanismos de composição e novos tipos de componentes, além de reduzir consideravelmente o número de dependências entre as classes. É o único paradigma que trabalha com este tipo de solução.

Em um sistema AOP [33,34,35] a estrutura principal de dados e algoritmos são modelados na tradicional POO com componentes e classes - denominados componentes normais. Os que requerem mecanismos de composições mais avançados - denominados aspectos do problema ou componentes aspectuais, ficam completamente independentes. O desenvolvimento é então facilitado por possibilitar a separação das responsabilidades transversais nestas unidades únicas (*aspectos)*, e a posterior composição junto as classe. Na fase final, o sistema é recomposto através de um processo denominado “weaving”.

O conceito da metodologia baseia-se no fato de que sistemas computacionais são melhores programados quando são especificadas separadamente todas as responsabilidades, propriedades ou áreas de interesse de um sistema em unidades modulares.

Apesar de AOP ser, inicialmente, mais complexo de se trabalhar do que os paradigmas tradicionais, o nível de modularidade que se alcança é elevado, o código é mais claro e de fácil manutenção. A complexidade é reduzida, visto que uma parte do código fica na definição dos aspectos, e por estarem centralizados em uma única unidade, alterações são mais simples, já que não é necessário reescrever inúmeras classes, o que reduz o impacto no desenvolvimento. AOP adiciona uma nova camada de solução ao desenvolvimento com as fases de decomposição, implementação e recomposição.

4.1 Conceitos Fundamentais

Neste item, os principais conceitos de AOP serão apresentados e uma rápida e inicial associação com o fluxo de potência será descrita.

- Responsabilidades (*concerns*)

Sistemas de software consistem de um conjunto de "áreas de interesse" ou *responsabilidades* distintas como, por exemplo, responsabilidades funcionais (lógica de negócio) e não-funcionais (desempenho, verificação de erros, etc.). Existem também preocupações inerentes ao desenvolvimento do sistema, como clareza, entendimento, manutenção, etc.

Em caso de um FOP pode-se facilmente identificar duas responsabilidades básicas; o sistema elétrico com todas as suas variáveis, funções e características e o sistema de otimização, que pode ser implementado utilizando diversas metodologias com características e efetividades muito diferentes dependendo da aplicação.

- Separação de responsabilidades (*separation of concerns*)

A melhor maneira de se projetar um sistema é através da separação de suas responsabilidades distintas de tal modo que podemos alterar/re-projetar cada uma sem que isto afete as demais partes do sistema. Sistemas computacionais são melhores programados quando são especificadas separadamente todas as responsabilidades em unidades modulares. Este conceito esta presente na POO com a introdução da classe. Porém algumas funcionalidades são necessárias em diversas partes do problema.

Desta forma, a implementação de um FOP utilizando POO demanda a divisão das responsabilidades descritas anteriormente que geram duas classes de responsabilidade; características do sistema elétrico e metodologias de otimização.

- Responsabilidades transversais (*crosscutting concerns*)

Em sistemas com alta complexidade, sempre existem *responsabilidades* de interesse comum que são utilizadas por vários módulos. Estas responsabilidades são difíceis de isolar já que são necessárias em vários pontos do código. Em POO, uma classe oferece uma boa maneira de se separar a maioria das responsabilidades funcionais, mas é bastante limitada quando se trata de *responsabilidades transversais*. Com a POO, os *crosscutting concerns* ou *responsabilidades transversais* ficam espalhados por vários módulos em pequenos trechos de código que são, em geral, repetitivos, resultando em sistemas difíceis de projetar, entender, implementar, manter e evoluir.

É justamente neste item que a POO apresenta restrições no desenvolvimento de um sistema de FPO. É impossível, com os mecanismos de montagem de sistemas presente em OO, e.g. herança, polimorfismo, etc., lidar de forma eficaz com as responsabilidades transversais. Um exemplo fácil de visualizar o problema de responsabilidades transversais são as variáveis de controle do sistema elétrico que precisam ser as mesmas utilizadas pelos métodos de otimização. Desta forma, alterações no sistema de potência necessariamente geram alterações na codificação dos métodos de otimização.

4.2 A Metodologia AOP

Desenvolver um sistema usando AOP é similar ao desenvolvimento de outros sistemas com outras metodologias. Devem-se identificar os métodos e as classes, porém, também os aspectos, implementá-los e formar o sistema final pela composição de todos os elementos.

Porém, a padronização é fundamental para este paradigma. Neste trabalho foram utilizadas interfaces específicas para conseguir a padronização. Interface define um contrato, onde qualquer classe ou estrutura segue a forma definida.

AOP segue as seguintes fases para o desenvolvimento:

- **Decomposição**: neste passo é necessário identificar as várias *responsabilidades* do sistema e classificá-las em dois tipos: comuns ou transversais. *Comuns* para os componentes normais da POO como as classes com características comuns do sistema e *transversais* para os métodos transversais, utilizados de forma entrelaçada em várias partes do código. Estes irão compor os aspectos.
- **Implementação**: esta fase é responsável pelo desenvolvimento dos componentes normais e os aspectos. É fundamental seguir a padronização definida pelas interfaces.
- **Recomposição**: fase responsável por gerar a solução final através da combinação dos componentes e aspectos usando um montador denominando *Weaver* (detalhado ainda neste capítulo).

4.2.1 Aspectos

AOP trabalha com um novo tipo de módulo, conhecido como aspecto. Estes são similares as classes, tem um tipo, podem conter métodos, propriedades, campos e tipos como membros, podem ser abstratos ou concretos. Portanto não tem construtor e não podem ser instanciados.

Os aspectos podem ser vistos como uma responsabilidade, cujas funcionalidades são ativadas por outras em situações múltiplas. Por exemplo, se uma responsabilidade não estiver separada dentro de um aspecto, teria de ser ativada diversas vezes, em múltiplos pontos dentro do código de um determinado sistema.

Os aspectos ficam em um único bloco de código, por isto sua manutenção é mais simples, a complexidade é reduzida e o entendimento facilitado. Além disso, o código das classes fica livre do relacionado às responsabilidades transversais, facilitando a reutilização em

diferentes contextos, e dependendo das necessidades da aplicação diferentes aspectos podem ser combinados.

A figura 4.1 apresenta uma comparação entre a programação POO e AOP. Note que na solução POO o sistema está entrelaçado, enquanto que na AOP isto não ocorre devido a utilização do aspecto.

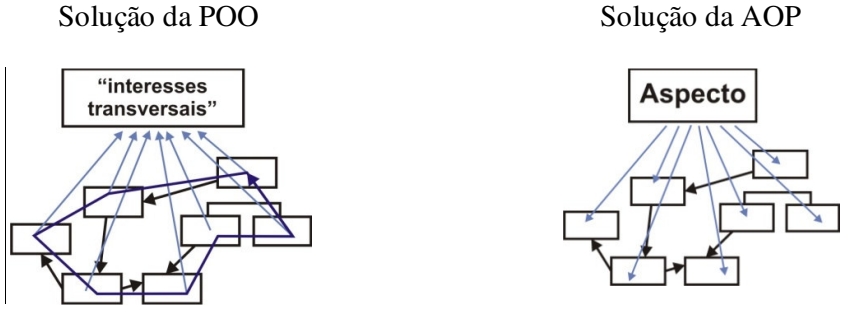

Figura 4.1 – Comparativo de projeto POO X AOP com aspecto

4.2.2 Weaver

Uma implementação básica de AOP consiste em: programar os componentes, programar os aspectos, e montar o sistema final através de um elemento capaz de carregar as instruções e compor os diferentes componentes juntos. O *Weaver* é responsável por esta composição.

Este processo, denominado *weaving* pode ser estático ou dinâmico. A ferramenta estática simplesmente leva os parâmetros especificando o componente (aspecto) assemblies nas linhas de comando, então aplica o procedimento de tecelagem a estes arquivos, criando um novo assembly. Já o objetivo do *weaver* dinâmico é dar suporte aos componentes em tempo de execução, para que ocorra, componentes devem ser interpretados quando assemblies dos aspectos forem referenciados por outros componentes funcionais do sistema.

Weaver Estático

- Weaver funde no código fonte antes da compilação.
- Weaver funde na IL (código intermediário gerado pelo compilador) antes da execução.

Weaver Dinâmico

- Biblioteca de Weaver é chamada com os aspectos e componentes como parâmetros.
- Weaver é chamado quando o componente é carregado para uso e os métodos são chamados.

O trabalho desenvolvido utilizou o Weaver dinâmico com capacidade de *reflection*. Reflection é um processo de acessar informações. Através deste consigo acessar qualquer, método, variável de qualquer classe do meu sistema. Inclusive acessar a classe. O tópico 4.5.2 trás mais informações deste processo.

4.3 Diferença entre Componentes Normais da POO e Aspectos.

Como já mencionado, AOP lida com um problema bem específico que é capturar conceitos de um sistema que ficam espalhados por diversas partes do sistema e quebrar os fortes relacionamentos existentes entre eles. Para isto, as *responsabilidades* são identificadas, quebradas, e o sistema dividido. Para efetuar esta divisão é necessário identificar os componentes normais da POO e os Aspectos.

O que diferencia um aspecto de um componente normal é a sua composição com o resto do sistema. Um componente normal pega um elemento do modelo e o subdivide utilizando relacionamentos internos e externos da tradicional POO. Um aspecto utiliza somente relacionamentos internos. Os externos só serão conhecidos quando o programa for executado e conforme a necessidade. A composição do sistema é finalizada pelo *Weaver*, responsável pela composição do sistema final.

A figura 4.2 exibe dois sistemas com a POO e AOP respectivamente. Os traços fortes indicam um forte relacionamento entre os componentes, e a linha tracejada indica ausência de um forte relacionamento. Observe que na figura 4.2(a), da POO, existe um forte relacionamento, representados pelos traços entre os componentes, sejam eles internos (dentro dos componentes "A", "B" e "C") ou externos (Componentes "A" com "B" e "B" com "C") . Isto significa que existe uma forte ligação entre os mesmos. Se for preciso alterar algum componente, necessariamente várias outras partes do sistema deverão também ser alteradas.

Já a figura 4.2(b) ilustra um sistema desenvolvido conforme a programação orientada a aspectos. A linha tracejada entre os módulos, neste caso representados pelos aspectos "A" "B" e "C", demonstra a ausência de um forte relacionamento externos entre os mesmos. Os relacionamentos são somente internos, e em cada aspecto em questão. Os externos só serão criados futuramente conforme a necessidade (linha tracejada na figura 4.2(b)).

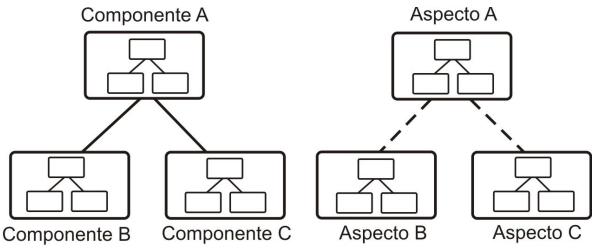

Figura 4.2 – a) sistema POO com componentes normais b) sistema AOP

Componentes: objetos de sistema. Possuem relacionamentos internos e externos.

- Aspectos: similar a componentes, porém só possuem relacionamentos internos.

Em AOP, ao quebrar os relacionamentos externos quebram-se as dependências e uma enorme economia de manutenção é alcançada. Neste modelo qualquer alteração em um aspecto não altera os demais, ou seja, se alterar ou inserir algum código, como por exemplo, inserir

algum novo método de otimização, somente o aspecto em questão será alterado, sem mexer em outras partes do sistema. O resultado é um sistema mais fácil de trabalhar com uma lógica mais complexa, além da economia de manutenção alcançada, já que o retrabalho é bem reduzido.

Para criar os relacionamentos externos o novo paradigma orientado a aspectos fornece três ferramentas fundamentais:

Reflection: Processo de acessar informações. É possível acessar qualquer, método, variável de qualquer classe do meu sistema. Inclusive consigo acessar a própria classe em tempo de execução.

Interface: Garante a padronização do sistema. Interfaces também estão presentes da POO, porém na AOP tem um papel fundamental de padronização, o que é extremamente necessário para a metodologia.

Weaver: O Weaver interpreta os aspectos, vê quais os componentes serão utilizados para montar o sistema final.

4.4 Como os Programas são Executados

A figura 4.3(a) mostra a execução do código em um programa convencional. A figura 4.3(b) representa a execução do código com o uso da programação orientada a aspectos. Nela são definidos alem do código geral do sistema o aspecto, sendo necessário o interpretador do aspecto (Weaver) para então o programa ser compilado gerando o executável.

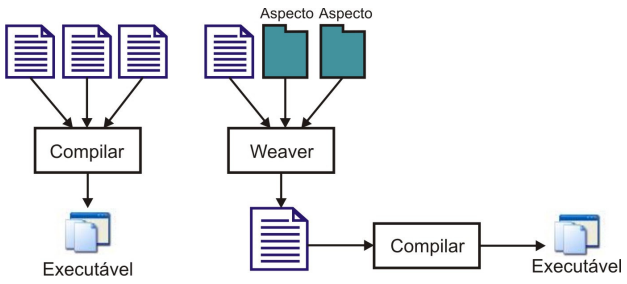

Figura 4.3 a)Programação convencional b) Programação orientada a aspectos

4.5 Principais Linguagens

4.5.1 AspectJ

AspectJ é uma linguagem muito utilizada para o desenvolvimento de programas com AOP. É uma extensão da linguagem JAVA que suporta a implementação de aspectos, separando assim as responsabilidades transversais em módulos. O AspectjJ adiciona novos tipos de elementos ao JAVA, como: aspectos, pointcuts e advices. Maiores informações em [35].

4.5.2 AOP em C#

Baseado na arquitetura. NET, a linguagem C# permite uma fácil implementação de programas orientados a aspectos. Os aspectos são construídos na estrutura ou na definição convencional de classe. Para fornecer serviços comuns a todos os aspectos, é criada uma interface para o mesmo. Interface em C# define um contrato, onde qualquer classe ou estrutura segue a forma definida. O Weaver identifica os aspectos interpretando-os. Para que seja possível esta identificação nos objetos, a linguagem usada possui *metadados* ou informações sobre seus dados, tais como campos, métodos, construtores e interfaces que o objeto implementa.

Reflection é a capacidade de consultar e usar estes *metadados*, ou seja, é um processo responsável por obter informações sobre *assemblies* e os tipos definidos dentro dele, e ainda criação, invocação e acesso às instâncias de tipos em tempo de execução. Reflection é, segundo a definição existente no .Net Framework SDK, o processo de obter informações sobre *assemblies* e os tipos definidos dentro dele, e ainda a criação, invocação e acesso às instâncias de tipos em tempo de execução.

Na programação com C# a utilização do *Weaver* pode ser tanto na forma estática como dinâmica. A estática simplesmente especifica o componente do aspecto. Na dinâmica o *Weaver* apóia os componentes em tempo de execução, na medida da necessidade. São referenciados

quando afetados por componentes funcionais do sistema. O *Weaver* utilizado no trabalho foi o dinâmico, com capacidade de Reflection.

4.6 AOP no Trabalho

Umas das áreas mais importantes em sistemas elétricos é o estudo de problemas de fluxo de potência ótimo. As áreas de aplicação variam de planejamento da expansão, operação e manutenção de uma rede elétrica até solucionar problemas no mercado de energia [1,2,4,3,5,9,10,11,12,13]. Dependendo do espaço de solução e dos tipos de variáveis o problema pode ser dividido em diversas categorias e utilizar diversos métodos ou conceitos de otimização para solucioná-los.

Um dos focos deste trabalho foi empregar o paradigma de modelagem orientado a aspetos de forma a melhorar o desenvolvimento e manutenção destes sistemas.

A figura 4.4 descreve o comportamento de um modelo genérico, onde, dentro deste sistema temos os métodos de otimização, os métodos numéricos, o fluxo de potência e entrelaçado a todos estes módulos tenho as variáveis, os objetivos e as funções.

Neste contexto, qualquer alteração em termos de variáveis ou objetivo, é necessária alterar toda esta estrutura. Sem mencionar os componentes que influenciam outros componentes. O entrelaçamento lógico existente é enorme, além da grande dificuldade de implementar várias possibilidades de configurações.

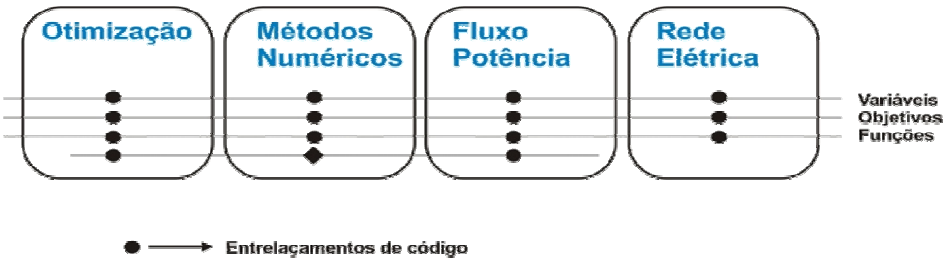

Figura 4.4 – Exemplo Sistema

Para um exemplo prático, imagine que o usuário deseja alterar/inserir algum método de otimização e utilizá-lo para resolver o problema. Como existe um forte relacionamento entre os componentes, além do módulo de otimização, diversas alterações em outros módulos são também necessárias, dificultando a manutenção e o entendimento.

Como o objetivo da AOP é quebrar estes fortes relacionamentos o desenvolvimento do FPO com AOP foi feito de forma independente. Ou seja, o fluxo de potência foi feito de forma completamente independente, o método de otimização foi desenvolvido também de forma totalmente independente. Só então a solução final é montada pelo Weaver.

Como os módulos são independentes nos aspectos, qualquer alteração é feita somente no módulo em questão. Inserir quaisquer novos métodos seja ele numérico ou inteligente é muito mais simples (já que não é necessário alterar todo o sistema). Desta forma, o entrelaçamento lógico é evitado.

O próximo capitulo aborda todo o desenvolvimento do FPO com AOP.

4.7 Benefícios da AOP

As críticas sobre AOP freqüentemente abordam a dificuldade do seu entendimento. Realmente para se tornar um especialista requer mais tempo, pratica e paciência. Porém, a principal razão por trás da dificuldade é simplesmente a renovação da metodologia. Entretanto os benefícios de AOP compensam toda dificuldade inicial. Dentre os principais benefícios estão:

- Maior reuso do código: como o código separa os interesses em unidades o reuso em sistemas diferentes é facilitado.
- Alta modularização: como há a separação dos interesses transversais o programa é mais modular, o que facilita também o entendimento e manutenção
- Fácil evolução do sistema: adicionar novas funcionalidades significa incluir um novo aspecto, sem necessidade de alterar o código existente.

5 Programação Orientada a Aspectos Desenvolvimento de FPO utilizando a Programação Orientada a Aspectos

O paradigma de desenvolvimento AOP é uma poderosa ferramenta de modelagem, porém requer mais tempo de análise e codificação. A utilização de aspectos em todas em camadas não é recomendada, já que consumiria muito tempo no desenvolvimento sem um ganho considerável. Portanto o uso do paradigma só é recomendado em situações onde exista um forte acoplamento entre alguns componentes, como no caso de um sistema de potência acoplado a um método de otimização. Em problemas de FPO é possível aplicar AOP em diversas partes, como em tratamento de erros, montagem do jacobiano, etc.

A seguir, com intuito de aplicar as melhores práticas, a modelagem de um FPO com AOP será apresentada. A análise inicial será fornecida pelo paradigma OO (Figura 3.4) onde todas as classes estão interconectadas através de dependências diretas ou indiretas. Isto significa que diversas funcionalidades de uma classe são utilizadas por outras, e se espalham por todo o sistema. Nestes casos, quando qualquer alteração em alguma funcionalidade é realizada, mudanças nos lugares em comum com outras são obrigatórias.

Para provar a diferença de AOP com a POO é necessário medir o impacto antes e depois da implementação. Existem diversas formas de se medir o impacto que uma classe A possui sobre uma segunda classe B. Neste trabalho esta medida foi definida como um índice heurístico,

representado pela probabilidade de A necessitar atualização devido a novas funcionalidades necessárias em "B".

Este índice mede os custos de manutenção considerando a probabilidade de ocorrência de uma mudança (e não considerando que irá acontecer). Por exemplo: duas classes tem uma conexão muito forte, entretanto um especialista aponta que elas nunca irão sofrer nenhum tipo de alteração. Neste caso o impacto é nulo. Por outro lado, se duas classes tem conexão fraca, mas são passíveis de mudanças periódicas, o impacto é alto.

De posse dos índices heurísticos, passados por especialistas em fluxo de potência, a Tabela 5-1 aponta os impactos entre as classes. Os índices vão de nulo, nenhum impacto, a 4, representando um impacto muito alto. Pela tabela é possível concluir que a classe que sofre maior impacto quando alguma mudança ocorre em outra classe é a ClsController.

Tabela 5-1- Imapacto de uma Solução POO

	ClsPowerFlow	ClsData	ClsBuses	ClsLines	ClsController	ClsOptMethods
ClsPowerFlow		1	1	1		
ClsData			1	1		
ClsBuses						
ClsLines						
ClsController	4	4	4	4		3
ClsOptMethods	2	2	2	2		

O primeiro passo para eliminar esta dependência é desenvolver os domínios do fluxo de potência completamente separados uns dos outros. O resultado serão aspectos completamente independentes, com diversos relacionamentos internos, mas nenhum externo.

A padronização é fundamental no desenvolvimento de um sistema com AOP. Isto se deve ao fato dos principais conceitos da orientação a aspectos serem baseados em *metadados* e

reflexão. Através da reflexão é possível ler e chamar, durante a execução do sistema, qualquer variável e função presente no objeto e que estão representados no *metadado*.

Com a utilização desta funcionalidade, aliada ao fato de que o usuário tem acesso ao *metadado* de cada componente, é possível criar um domínio genérico, representado pelo Weaver. Para melhor compreensão, considere a classe *ClsBuses* que armazena em um objeto *objBus*, todas as variáveis, propriedades e métodos de uma barra. Agora suponha, por exemplo, que é necessário analisar a geração ativa da barra, representada pela variável *cPg*, que está presente em objeto *objBus*. Com a POO seria necessário ter pré-implementado um tratamento lógico para identificar o campo desejado e retornar seu valor. Entretanto, utilizando reflexão é possível acessar este valor através do *metadado* do objeto, como mostra o código abaixo.

Algoritmo 1 *Chamada de campo através de reflexão*

```
public class ClsBus {
public double C_Pg; // active generation
public double S_Teta;// angle
…
public double FieldValue(string fieldName)
{   1 Type myType = this.GetType();
    2 FieldInfo Minf = myType.GetField(fieldName);
    3 return (double)Minf.GetValue(this);}}
```

A reflexão, apresentada acima, funciona da seguinte forma:

Tag 1 o tipo específico do objeto é identificado. Neste exemplo o tipo é *ClsBus*.

Tag 2 o *metadato* de um campo desejado é recuperado e armazenado em *Minf*. Este campo representa a meta-informação da variável.

Tag 3 a *metainformação* é utilizada para recuperar o valor do campo específico.

Um ponto importante da metodologia é que os nomes dos elementos do problema de fluxo de potência (variáveis de controle, estado e funções) são variáveis, ou seja, não são conhecidas pelo sistema até a execução da rotina. Com a utilização destas meta-variáveis, ao invés dos nomes reais das variáveis, é possível evitar a complexa lógica de tratamento do problema.

Porém, o grande problema desta metodologia está na padronização. Durante a execução do sistema o usuário tem que visualizar quais as variáveis disponíveis, e então definir uma estratégia de controle. Logo, o sistema de fluxo de potência e o de otimização devem fornecer uma lista das variáveis e funções disponíveis para que o usuário monte o problema que lhe for conveniente.

Utilizando reflexão, o Weaver lê a meta informação contida nos componentes e mostra ao usuário. Continuando com o fluxo do funcionamento do sistema, o usuário escolhe a estrutura do FPO para gerar a solução final.

Para identificar as variáveis do problema é indispensável uma assinatura específica. Existem diversas maneiras de se criar estas assinaturas. Neste trabalho foram utilizadas interfaces específicas para implementar os métodos, funções e variáveis.

5.1 Desenvolvimento dos Aspectos

As Figuras 5.1 e 5.2 mostram como as interfaces são utilizadas na implementação do sistema. Em relação ao domínio do fluxo de potência, todas as funções presentes utilizam a interface *IFunctions*, as variáveis de barra usam a interface *IBusVariabel* e as de linha a *ILinesVariables*. Qualquer nova funcionalidade, realizada a partir das interfaces, é automaticamente absorvida pelo *Weaver* e incorporada ao problema.

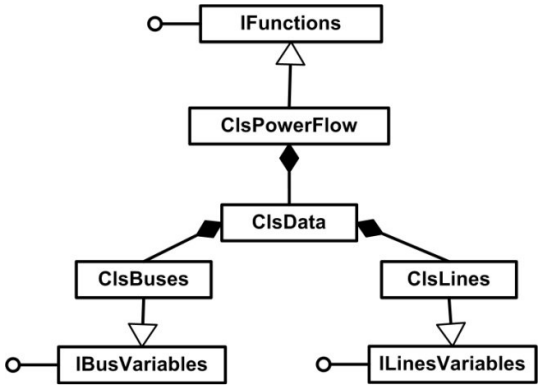

Figura 5.1- Aspecto do Fluxo de Potência

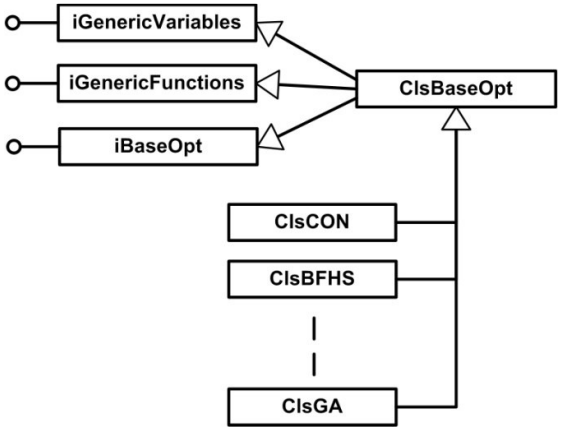

Figura 5.2- Aspecto de Otimização

Para aplicar esta técnica em um problema de FPO foi desenvolvido um componente *Weaver* conforme mostrado na Figura 5.3. A solução possui duas classes, *ClsGeneric* e *ClsIntelligentOptz*, cada uma responsável por acessar a meta informação de suas respectivas interfaces. As conexões pontilhadas da figura apontam a inexistência de conexões fortes entre estes componentes. As classes estão preparadas para acessar apenas as *meta-informações* contidas nas suas interfaces, assim qualquer mudança no sistema de otimização ou de fluxo de potência, não acarretará em alterações do código nestas classes.

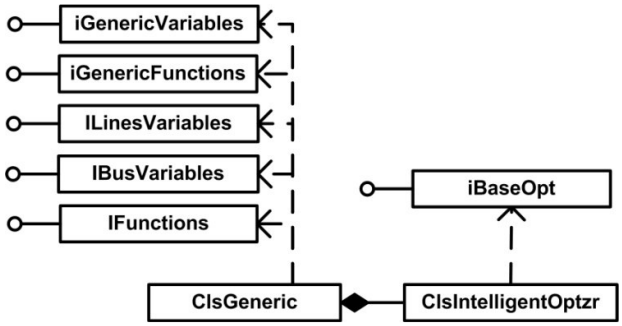

Figura 5.3- Modelo Final da Solução AOP

Através deste acesso, o sistema AOP é capaz de exibir todas as informações úteis sobre variáveis, funções e métodos de otimização presentes nos aspectos, permitindo ao usuário ajustar e executar uma configuração desejada do FPO;

A solução final tem três componentes, dois relacionados aos domínios de interesse e o terceiro de composição: fluxo de potência (Figura 5.1), otimização (Figura 5.2) e o Weaver (Figura 5.3). O componente de fluxo de potência contém todo o conhecimento referente ao sistema elétrico de potência. O componente de otimização contém o conhecimento abstrato dos métodos de otimização e processos sem fazer referência a uma variável específica ou função objetivo. Durante o desenvolvimento não existem conexões entre estes componentes. O componente Weaver encapsula as informações de como o componente do fluxo de potência e o de otimização irão se conectar.

Os *metadados* pertencentes às variáveis, funções, e métodos de otimização estão representados no componente *Weaver* (*ClsGeneric* e *ClsIntelligentOptzr*). São estes *metadados* que o sistema irá utilizar durante sua execução. Para relacionar estes com as variáveis reais do sistema, o AOP executa uma sobrecarrega dos dados reais sobre os genéricos. Desta forma, é possível ter acesso a qualquer variável real utilizando a genérica através do método de reflexão. Sendo assim, durante o tempo de execução, o usuário ajusta da forma desejada o sistema que será otimizado.

Com esta solução é possível separar totalmente o domínio do fluxo de potência do de otimização. O grafo da solução é mostrado pela Figura 5.4.

Figura 5.4- (a) Seleção da Função Objetivo, (b) Seleção das Variáveis de Controle

O tratamento lógico já não é mais parte da solução, o número de funções objetivo, variáveis de controle não implicam em aumento de complexidade. O paradigma possibilita a completa separação dos aspectos e, mudanças em determinadas partes não impactam as demais.

A Figura 5.5 apresenta de modo simplificado o gráfico da solução, onde as interfaces indicam a forma que as funções, variáveis e métodos de otimização devem possuir. O Weaver representa um tabuleiro pré-especificado, com encaixe para todos os componentes que possuam a assinatura correta.

Desta forma, é possível escolher qual a funcionalidade desejada e ligá-la através do Weaver. Neste exemplo, um problema de redução de sobrecarga utilizando atração dos taps de transformadores foi selecionado e, como ferramenta de otimização, o método BFHS foi escolhido.

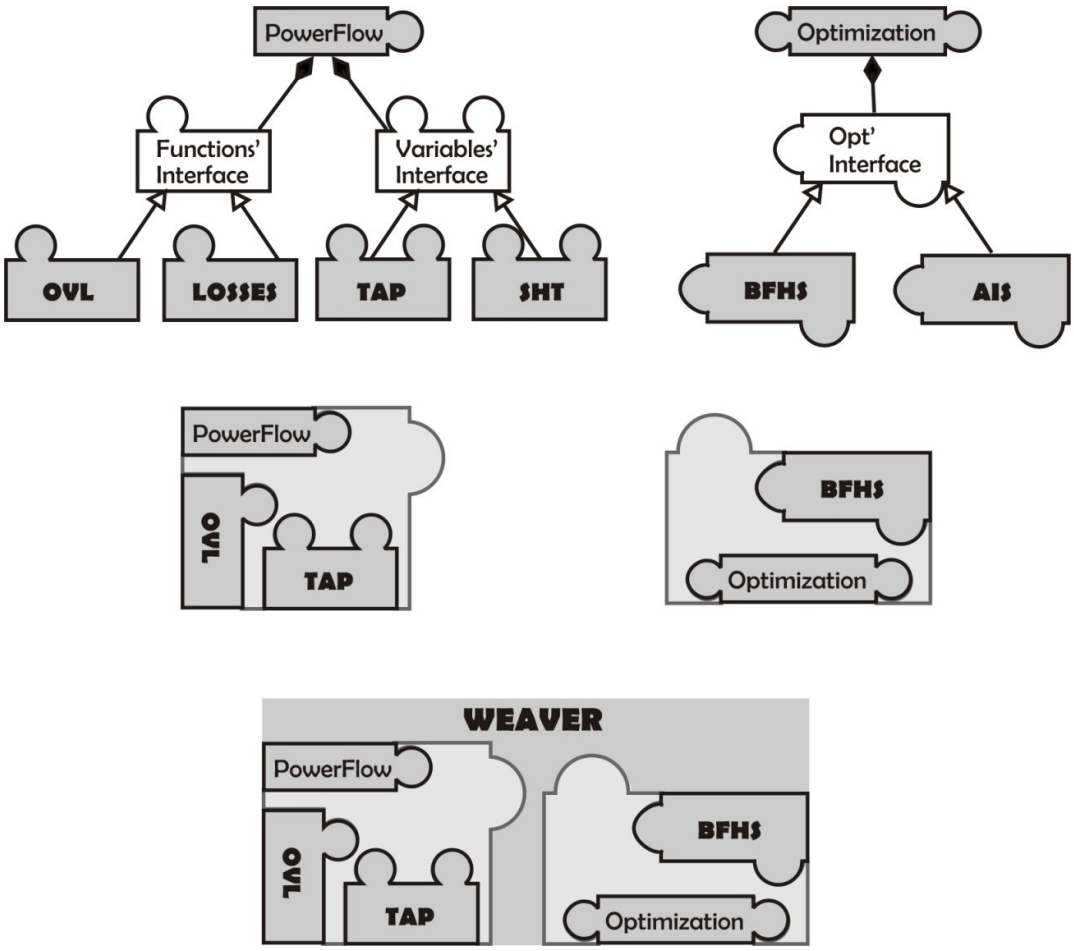

Figura 5.5- Gráfico Informativo da Solução AOP-FPO

Na figura 5.5 o módulo responsável pelo Fluxo de Potência é representado por *PowerFlow*, o da otimização por *Optimization*. O *Weaver* é responsável por compor o sistema final através dos Aspectos com as funções, variáveis e método de otimização escolhidos.

Funciona da seguinte forma: *PowerFlow* é um componente ligado as funções genéricas e variáveis genéricas. Cada variável ou função genérica segue a forma de sua interface, e de acordo com a necessidade cada uma é solicitada. O aspecto é então montado conforme as escolhas. Na figura, para o fluxo de potência foi escolhido sobrecarga (OVL) com TAP . Para a otimização o princípio é o mesmo. Temos o componente *Optimization* ligado aos métodos de

otimização genéricos que seguem a interface *Opt`Interface*. Para montar o aspecto basta escolher qual método será utilizado. Ex: método de otimização BHFS.

Todas estas funções genéricas (losses, ovl), variáveis genéricas (Tap, sht) e métodos de otimização genéricos (BHFS, AIS) têm uma assinatura. A sobrecarga dos valores reais aos genéricos é feita através do *Reflection.* Através deste processo o sistema pega a variável genérica e sobrecarrega com a real. A função real está implementada dentro do *PowerFlow*. O mesmo ocorre para as variáveis, tem-se a genérica, sabe-se qual o comportamento dela, e através do reflection é feita a sobrecarga da genérica com a real. Os métodos de otimização seguem o mesmo princípio.

A grande vantagem é que possível identificar pelo *reflection* qualquer variável inserida. O sistema não precisa ser informado que a mesma foi adicionada. O mesmo para qualquer nova função objetivo. Por *reflection* é possível ler e identificar que esta nova função agora existe e então mostrar ao operador. Com AOP as alterações necessárias serão só locais, não é necessário alterar a estrutura, nem trabalhar com lógica complexa alguma. O sistema vai lendo as funções e disponibilizando ao operador que escolherá a função e então sobrecarregar a interface. O resultado final são dois aspectos, um para o fluxo de potência e outro para o método de otimização.

O Weaver monta o sistema final pela composição dos dois aspectos, utilizando não mais funções e variáveis genéricas, e sim já sobrecarregadas (No exemplo Minimizar Sobrecarga com método BHFS e ação de controle TAP). Depois desta sobrecarga, quando o Weaver chama a função que sobrecarregou com perdas, na verdade chama por *Perdas*. Com tudo sobrecarregado a solução final é modular, versátil e montada de acordo com a necessidade do operador.

5.2 Impacto no Desenvolvimento com AOP

A Tabela 5-2 mostra uma tabela de impacto para o desenvolvimento com AOP. É possível concluir que a metodologia AOP quebra todos os relacionamentos externos entre os componentes, preservando apenas os internos. Mesmo que algumas conexões ainda existam, elas não possuem impacto sobre outras classes.

Tabela 5-2 Tabela Impacto da Solução

		ClsPowerFlow	iFunctions	ClsData	ClsBuses	ClsLines	iBusVariables	iLinesVariables	ClsBaseOpt	ClsOptMethods	iGenericVariables	iGeneric Functions	iBaseOpt	ClsIntelligentOptzr	ClsGeneric
Power Flow Aspect	ClsPowerFlow		2	1			1	1							
	iFunctions						1	1							
	ClsData						1	1							
	ClsBuses						1								
	ClsLines							1							
	iBusVariables														
	iLinesVariables														
Optimization Aspect	ClsBaseOpt												2		
	ClsOptMethods										2	2	2		
	iGenericVariables														
	iGeneric Functions										1				
	iBaseOpt										2	2			
Wvr	ClsIntelligentOptzr						0	0			0	0	0		4
	ClsGeneric						0	0			0	0	0	4	

A solução final pode ser visualizada na Figura 5.6. O processo começa com o *Weaver* lendo o metadado presente nos aspectos de fluxo de potência e de otimização. As funções, variáveis, etc, são então apresentadas para o usuário que é responsável por definir quais as ligações que serão realizadas. O *Weaver*, de posse desta informação, gera todas as conexões necessárias para criar o sistema de fluxo de potência ótimo selecionado pelo usuário.

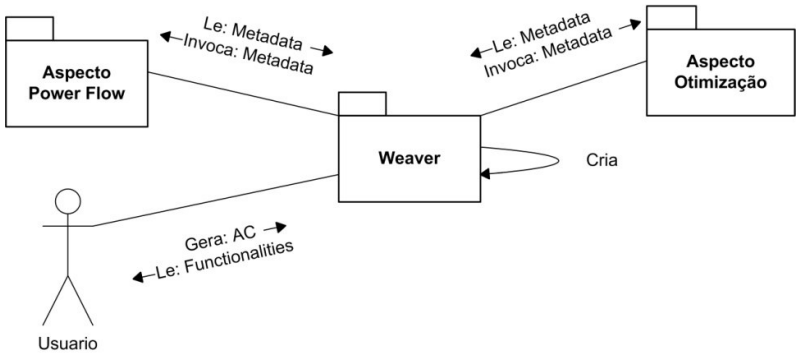

Figura 5.6- Diagrama UML da Solução final AOP - FPO

A consideração final de uma solução baseada em AOP é que o tempo computacional e a precisão são os mesmos obtidos por um sistema desenvolvido inteiramente por OOP. Existem três razões para isto: A) o aspecto de otimização pode ser desenvolvido utilizando qualquer algoritmo e a qualidade do resultado depende apenas disso, B) o processo de "weaving" é executado apenas no início de cada simulação e requer muito menos esforço computacional do que os exigidos em um fluxo de potência. Logo, o tempo final da solução é exatamente o mesmo, C) AOP é apenas um paradigma de desenvolvimento e trabalha visando melhorar o entendimento e o desenvolvimento de um sistema com um nível maior de abstração. Assim, o resultado numérico não é afetado.

6 Sistema Imunológico Artificial Baseado em Clusters e Gradiente

Como já mencionado, este trabalho tem dois focos: facilitar o desenvolvimento de sistemas, evitar retrabalho e melhorar a interação com o operador através do paradigma orientado a aspectos (capítulos anteriores); e aprimorar um algoritmo baseado no sistema imunológico artificial com o objetivo de aumentar a precisão e reduzir o tempo computacional.

Este capítulo apresenta os principais conceitos do sistema imunológico natural, do artificial, o novo algoritmo, as estratégicas utilizadas para alcançar o objetivo, como: clusterização, gradiente e alfa- Constrained, e validação e testes do algoritmo.

6.1 O Sistema Imunológico

A analogia entre sistemas biológicos e sistemas computacionais está cada vez mais explorada por pesquisadores que, motivados pela alta sofisticação e eficácia dos sistemas biológicos utiliza-os como fonte de inspiração para as mais diversas aplicações.

A metodologia do sistema imunológico artificial (AIS) é uma destas técnicas. Tem como princípio o sistema imunológico natural. As características de um sistema imunológico podem ser utilizadas principalmente para reconhecimento de padrões, otimização e aprendizado.

Os próximos tópicos têm como objetivo apresentar uma breve descrição do funcionamento do AIS, analisá-lo e finalmente demonstrar a aplicação do método.

6.2 O Sistema Imunológico Natural

Todos os organismos vivos são expostos a componentes do ambiente e sofrem constantes invasões de agentes infecciosos. Conseqüentemente, os organismos multicelulares desenvolveram diversos mecanismos de defesa que são desencadeados em decorrência de infecções e levam a destruição dos agentes agressores e seus fatores de virulência. De maneira geral, os mecanismos que compõem o sistema imune agem na vigilância, na iniciação, na manutenção de respostas protetoras, e no restabelecimento da homeostase (processo através do qual um organismo mantém as condições internas constantes necessárias para a vida) do organismo.

O sistema imunológico natural é um sistema complexo responsável por defender o corpo combatendo disfunções de suas próprias células e, de microorganismos invasores, conhecidos como elementos não próprios. Dois sistemas são responsáveis pelo reconhecimento destes microorganismos: o sistema imune inato e o sistema imune adaptativo.

O sistema imune inato constitui a primeira linha de defesa contra muitos microorganismos comuns. É formado por células fagocitárias, eosinófilos e várias moléculas originadas no sangue. Todos estes componentes defendem o corpo de um ambiente potencialmente hostil, e estão presentes antes da exposição a microorganismos infecciosos ou moléculas estranhas. Estes, não são potencializados por tais exposições e não discriminam entre a maioria das substâncias estranhas.

Já no sistema imune adaptativo outros mecanismos de defesa são induzidos ou estimulados pela exposição a substâncias estranhas, são peculiarmente específicos para moléculas distintas e aumentam em magnitude a capacidade defensiva para cada exposição sucessiva a uma molécula em particular. As principais células deste sistema são os linfócitos e seus produtos, como os anticorpos, que são secretados pelos linfócitos B efetores, os plasmócitos. As substâncias estranhas que induzem a imunidade específica são denominadas antígenos. Na imunidade adaptativa, quando os linfócitos entram em contato com o antígeno, se

proliferam e passam a ter a capacidade de gerar clones idênticos que irão aumentar o número de células que apresentam receptores específicos para combater o agente que desencadeou a resposta. Quando a resposta mediada pelos linfócitos B é dirigida aos antígenos protéicos, existe a necessidade de colaboração com os linfócitos T. Após esta interação, o linfócito B que secretará o anticorpo passa por um processo de proliferação, fase conhecida como expansão clonal, ao final, se diferencia em plasmócito que poderá então produzir e secretar os anticorpos.

Durante a expansão clonal, uma parte dos linfócitos que estão em proliferação não se diferencia em células efetoras, se diferencia em células de memória, e ficam circulando pelo organismo. Estas serão ativadas quando o organismo entrar em contato pela segunda vez com o antígeno.

Também durante a expansão clonal, ocorre um processo importante que é o aumento da afinidade de interação entre o receptor ou anticorpo e o antígeno protéico. Nesta fase, o repertório de reconhecimento antigênico dos linfócitos pode passar por um processo de maturação devido a existência de dois mecanismos: hipermutação somática e edição de receptores. Através da troca das bases nucleotídicas no DNA, a hipermutação somática permite refinar a resposta imunológica ao antígeno reconhecido, modificando os genes de moléculas de imunoglobulinas e gerando produtos. Durante a edição de receptores, são capazes de reconhecer com maior afinidade aquele antígeno. Porém, durante estes processos podem ser criadas células auto-reativas (células capazes de produzir anticorpos que reconhecem antígenos próprios). Estas células B que apresentam receptores auto-reativos são eliminadas e aquelas que não reconhecem componentes próprios são preservadas.

A atuação conjunta dos mecanismos de geração e diversificação dos anticorpos (hipermutação somática e edição de receptores) faz com que o sistema imunológico seja capaz de produzir uma quantidade quase infinita de receptores celulares distintos partindo de um genoma finito, sendo capaz também de eliminar as células que podem causar dano ao organismo.

O processo de maturação por afinidade é um processo Darwiniano de variação e seleção, conforme ilustrado pela Figura 6.1.

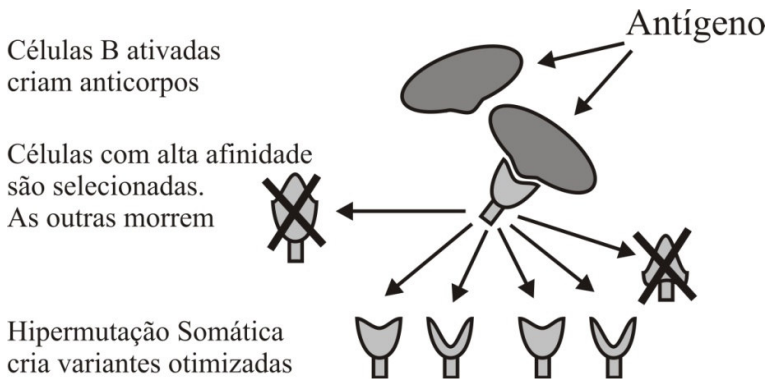

Figura 6.1- Processo de Maturação por Afinidade

Resumidamente: os linfócitos B que reconhecem o antígeno passam por uma fase que é a expansão clonal para aumentar o seu número. Nesta fase, as células que estão proliferando podem "melhorar" a capacidade de interagir com o antígeno. O DNA destas células, que contém os genes para produzir os anticorpos é fragmentado e pode se reorganizar para gerar moléculas que se ligam ao antígeno com maior eficácia. Os processos que resultam nisso são a hipermutação somática (reorganizar os pedaços do gene do anticorpo) e a edição de receptores (gerar moléculas que serão lidas e virarão as proteínas que são os anticorpos). Nem toda célula que reorganizar seu gene pode ser benéfica ao organismo. As indesejáveis (auto-reativas) serão eliminadas e as outras preservadas. As que foram preservadas, ao fim da proliferação, se diferenciam em efetoras (são os plasmócitos) ou de memória.

6.3 O Sistema Imunológico Artificial

O sistema imunológico artificial (AIS) é baseado no princípio do sistema imunológico natural. As características de um sistema imunológico podem ser utilizadas tanto para aprendizado como para otimização [21,36,37]. Para problemas de otimização três tópicos são interessantes: proliferação, mutação e seleção. Enquanto a proliferação é a capacidade de gerar novos indivíduos, mantendo a diversificação e fazendo o processo dinâmico de otimização, mutação é a habilidade de pesquisar dentro de um espaço de solução por pontos sub-ótimos. A mutação ocorre com duas características diferentes: hipermutação, responsável pela pesquisa local, e receptor de edição, responsável pela pesquisa global. O processo de seleção é responsável pela eliminação das células com baixa afinidade. Estas características fazem de AIS uma poderosa ferramenta que possibilita a pesquisa para vários ótimos locais.

Existem várias metodologias de AIS disponíveis na literatura para algoritmos de otimização. As referências [11,21,38] mostram uma estratégia interessante encontrada em alguns algoritmos evolutivos denominada agrupamento, onde indivíduos são direcionados a pontos mais promissores no espaço de solução. Apesar destes algoritmos apresentarem bons resultados o número de indivíduos usados no processo é alto, inadequado para problemas em grande escala e não-lineares.

Para incrementar os algoritmos originais CLONALG [21] e opt-AINnet [39], descritos ainda neste capítulo, e melhorar a dinâmica de problemas não-lineares e em grande escala serão realizadas duas modificações:

Na primeira, informações numéricas são inseridas ao gradiente durante o processo de hipermutação [11]. Desta forma é possível guiar este procedimento (no algoritmo original de AIS a geração dos melhores indivíduos era totalmente aleatória), logo o processo de otimização será mais rápido e confiável. Outra vantagem está na possibilidade de criar um critério de parada bem definido para cada anticorpo, desde que um gradiente nulo signifique um ponto estacionário.

Apesar deste método ter uma convergência mais rápida, o número de indivíduos definidos no início do processo é o mesmo. Como na metodologia AIS baseada em agrupamento todos os anticorpos são influenciados pelo mesmo atrator (um ótimo local) os indivíduos poderão convergir para um único ponto, gerando redundância. Baseado nesta informação a segunda mudança está na adição do controle de maturação ao algoritmo.

Com o uso destes conceitos pretende-se identificar os clusters de indivíduos influenciados pelo mesmo atrator e então, durante o processo de evolução eliminar os redundantes, exceto o melhor, acelerando a convergência.

Embora o conceito de clusterização tenha sido introduzido em [11], a estratégia adotada aqui é outra. A principal diferença está na redução da população durante o processo de convergência, diminuindo o esforço computacional.

Para demonstrar a eficácia do algoritmo proposto AIS baseado em gradiente e cluster (CGbAIS) serão analisados vários casos: otimização discreta, contínua e com restrições.

6.4 Principais Algoritmos

Dentre os algoritmos bem sucedidos, baseados em sistemas imunológicos, serão citados o CLONALG e Opt_AiNet.

6.4.1 CLONALG

O algoritmo de seleção clonal (CLONALG) foi proposto por [21]. Originalmente, foi desenvolvido para resolver problemas de aprendizagem de máquina e reconhecimento de padrões. Possui uma população aleatória de anticorpos que devem aprender e reconhecer um conjunto de antígenos. Devido às características adaptativas, foi estendido para aplicações de problemas de otimização.

O algoritmo visa representar a resposta da imunização adaptativa de um organismo na presença de um antígeno, reproduzindo aqueles indivíduos com altas afinidades e selecionando seus melhores clones. Através destes conceitos é possível gerar soluções de alta qualidade para problemas complexos. Utiliza o operador mutação como critério de diversificação e o operador clonagem como critério de intensificação. É composto por um conjunto de antígenos Ag e um conjunto de anticorpos Ab. O conjunto Ab pode se decompor em vários conjuntos de acordo com a aplicação.

A seguir a descrição genérica e simplificada do algoritmo de acordo com [21]:

Algoritmo *CLONALG*

1. Gerar um conjunto (P) de candidatos a solução;
2. Determinar (processo de seleção) os n melhores indivíduos (P{n}) da população (P),baseado em uma medida de afinidade;
3. Reproduzir (processo de clonagem) estes n melhores indivíduos, gerando uma população temporária de clones (C). A quantidade de filhos de cada indivíduo é diretamente proporcional à sua afinidade;
4. Submeter a população de clones a um esquema de hipermutação em que a taxa de mutação é proporcional à afinidade do anticorpo. Uma população de anticorpos maduros é gerada (C*);
5. Re-selecionar os melhores indivíduos de C* para compor o conjunto de memória M;
6. Substituir os d anticorpos por novos indivíduos (diversidade). Os anticorpos com menores afinidades possuem maiores probabilidades de serem substituídos.

Neste algoritmo, se n = N, ou seja, se todos os indivíduos da população forem selecionados para reprodução, cada candidato à solução será visto localmente, resultando em um algoritmo capaz de executar uma busca multimodal dentro do espaço de formas S.

As etapas 2 e 5 do CLONALG podem ser feitas de forma probabilística, ou seja, aqueles indivíduos com maiores afinidades terão maiores probabilidades de serem selecionados.

A literatura dispõe de outros algoritmos baseado na teoria da seleção clonal para as mais variadas aplicações.

6.4.2 OPT-AINET

Algoritmo inspirado no princípio da seleção clonal e teoria da rede imunológica que foi desenvolvido para solucionar problemas de otimização multimodal. É considerado um procedimento de busca evolutivo baseado na mutação e população com tamanho dinâmico. Apresenta várias características interessantes, como a possibilidade de ajustar o tamanho da população dinamicamente, exploração no espaço de busca para soluções novas e melhores, capacidade de localizar múltiplos ótimos, definir critérios de parada, dentre outros.

O algoritmo pode ser considerado uma extensão do CLONALG com passos envolvendo as interações anticorpos-anticorpo e não apenas interações antígenos-anticorpos. Possuiu um ciclo de maturação por afinidade e é repetido até atingir a estabilidade. No caso do Opt-Ainet a estabilidade da rede é medida pela diferença entre a afinidade média da população atual com a anterior.

O algoritmo inicia com geração aleatória e avaliação da população inicial. O critério de parada é guiado pela variação do tamanho da população sendo interrompido quando o tamanho da mesma não varia entre duas iterações consecutivas.

Segundo [39] o algoritmo segue os passos:

Algoritmo *OPT-AINET*

1. Inicar a população (o número inicial não é relevante);
2. Equanto não [população constante da memória], faça;
3. Calcular fitness (valor da função objetivo) e gere clones de cada célula da rede;

4. Mudar clones proporcionalmente ao fitness e determinar o fitness novamente;
5. Calcular a média fitness;
6. Se média fitness não variar, então continue. Senão, retorne ao passo 2;
7. Calcular a afinidade entre as células e eliminar todas, com exceção das que a afinidade seja menor que o limite s. Determinar o número após a eliminação;
8. Introduzir a porcentagem de células geradas aleatoriamente e retornar ao passo 2;
9. Fim Enquanto.

Este algoritmo gera um número fixo de clones de cada célula (anticorpo) da rede, sendo que este número não é proporcional a afinidade da célula. Posteriormente ocorre a hipermutação, geralmente aplicada a taxas inversamente proporcionais a afinidade de cada clone.

Os clones de uma célula formam uma sub-população, e a melhor célula é selecionada. Caso sua afinidade seja maior, esta substitui uma célula pai.

Quando a estabilidade é alcançada, ocorre uma etapa de supressão, onde se tenta eliminar a redundância de células na população.

6.5 CGbAIS: Sistema Imunológico Artificial Baseado em Gradiente e Cluster

No método AIS, o conjunto de variáveis de entrada ou de controle representa os anticorpos (Ab's). Em sistemas não-lineares o gradiente representa a sensibilidade do sistema, dada uma pequena perturbação ao redor de determinado ponto de operação. Estas sensibilidades são avaliadas para cada variável de controle e o resultado final define um vetor que indica a direção mais provável para alcançar o objetivo da otimização. Os incrementos aplicados geram os clones hipermutados.

6.5.1 Análise do Gradiente em Otimização Contínua

Existem vários caminhos para avaliar o gradiente. A escolha depende das características do sistema. Por exemplo, se um problema não-linear contínuo é utilizado, o vetor tangente ou o jacobiano (VT) podem ser aplicados. O VT trata a função objetivo como uma "caixa preta", onde pequenas perturbações são individualmente aplicadas sob cada dimensão do vetor de entrada, gerando o vetor indicado na equação 6.1

$$TV = \begin{bmatrix} \dfrac{f(x_1 + \Delta x_1, \ldots, x_n) - f(x_1, \ldots, x_n)}{|\Delta x_1|} \\ \vdots \\ \dfrac{f(x_1, \ldots, x_n + \Delta x_n) - f(x_1, \ldots, x_n)}{|\Delta x_n|} \end{bmatrix} \tag{6.1}$$

onde: n é o número de variáveis de controle ou de entrada, f(.) é a função objetivo a ser otimizada; $x_1,\ldots,x_n$ são as variáveis de entrada; e Δx_k é o incremento aleatório limitado para x_k.

No caso de VT, o número de clones hipermutados (nc) deve ser igual ao número de variáveis de entrada, por exemplo, n, desde que este número seja suficiente para assegurar que a sensibilidade de todo o espaço seja capturada apropriadamente.

Como exemplo, considere a função apresentada em [21], e exibida pela equação 6.2

$$Max \;\; f(x_1, x_2) = x_1 \sin(4\pi x_1) - x_2 \sin(4\pi x_2 + \pi) + 1 \tag{6.2}$$

O vetor tangente (VT) é apresentado pela equação 6.3:

$$TV = \begin{bmatrix} \dfrac{f(x_1 + \Delta x_1, x_2) - f(x_1, x_2)}{|\Delta x_1|} \\ \dfrac{f(x_1, x_2 + \Delta x_2) - f(x_1, x_2)}{|\Delta x_2|} \end{bmatrix} \tag{6.3}$$

Aplicar algum destes vetores no processo de hipermutação significa gerar o seguinte clone:

$$cAb_i = Ab_i + \alpha \times rand \times Gr \tag{6.4}$$

onde, cAb_i é o clone do elemento i, α é a mutação dada pela direção do gradiente, *rand* é um número aleatório entre de 0 e 1, e Gr é o vetor gradiente, calculado pelo VT. Na equação, "$\alpha.rand.Gr$", representa o passo de mutação empregado.

Como o gradiente tende a ser nulo perto de um ponto estacionário, o valor médio da expressão diminui, a precisão numérica aumenta e saltos ao redor do ótimo local são evitados. Porém, para aumentar ainda mais a precisão numérica e possibilitar um critério de parada bem definido, o processo GbHipermutação é utilizado e descrito pelo algoritmo *lista 1*.

A seguir a descrição da terminologia é apresentada.

Ab{nps}{n} estrutura representando a população composta por nps anticorpos e n variáveis de entrada.

Ab{i}{j} campo representando a variável de entrada j do anticorpo i;

Ab{i}.gr campo de dimensão n, representando o vetor gradiente de Ab{i};

Ab{i}.open Campo booleano, representando se o anticorpo ainda está aberto para mutações ou não;

Ab{i}.α fator de mutação;

Ab{i}.fit valor da função de avaliação;

F(Ab{i}) função de avaliação;

cAb{i} clone de Ab{i}, herda todos os seus campos;

β parâmetro para controlar a mutação em caso de falha na melhoria;

α_{min} comprimento de passo mínimo.

Algoritmo *Lista 1 – Processo de Hipermutação com entrada Ab{nps}{n}*

```
FOR EACH Ab{i} IN Ab{nps}
   IF1 Ab{i}.open == TRUE
   Ab{i}.gr = GRDVECTOR(Ab{i});
   cAb{i}=Ab{i}+Ab{i}. α* rand* Ab{i}.gr;
   cAb{i}.fit = F(cAb{i});
   IF2 (cAb{i}.fit > Ab{i}.fit)
      Ab{i}.fit = cAb{i}.fit;
   ELSE2
   Ab{i}.α = β* Ab{i}.α};
      IF3 (Ab{i}.α < α_{min} )
      Ab{i}.open = FALSE;
      END IF3
   END IF2
   END IF1
END FOR EACH
```

A função GRDVECTOR(.) pode ser calculada pelo vetor tangente. É possível concluir que VT precisa de 3 clones (um para cada dimensão e outro representando a direção conjugada final).

Baseado no sistema dado pela equação 6.2, uma análise sobre a eficiência do algoritmo é ilustrada pela Figura 6.2, onde todos os Ab's com todos os clones são apresentados. As figuras 6.2(a), (b) e (c) exibem a técnica VT e, as figuras 6.2(d), (e) e (f) o método utilizado pelo CLONALG e opt-AiNET. É importante salientar que estes dois algoritmos possuem basicamente o mesmo processo de hipermutação, logo, os resultados são bem parecidos.

Analisando os resultados obtidos pelo CLONALG, com as melhorias fornecidas pela adição da informação do gradiente ao processo de convergência, é possível concluir que a alteração trouxe grandes vantagens ao processo evolutivo. Prosseguindo com a análise e considerando que cada ponto da figura é uma avaliação do problema, pode-se concluir também que sistemas com alta dimensão, como sistemas elétricos que alcançam facilmente milhares de variáveis, teriam uma velocidade maior no processo de convergência, conforme a Figura 6.2.

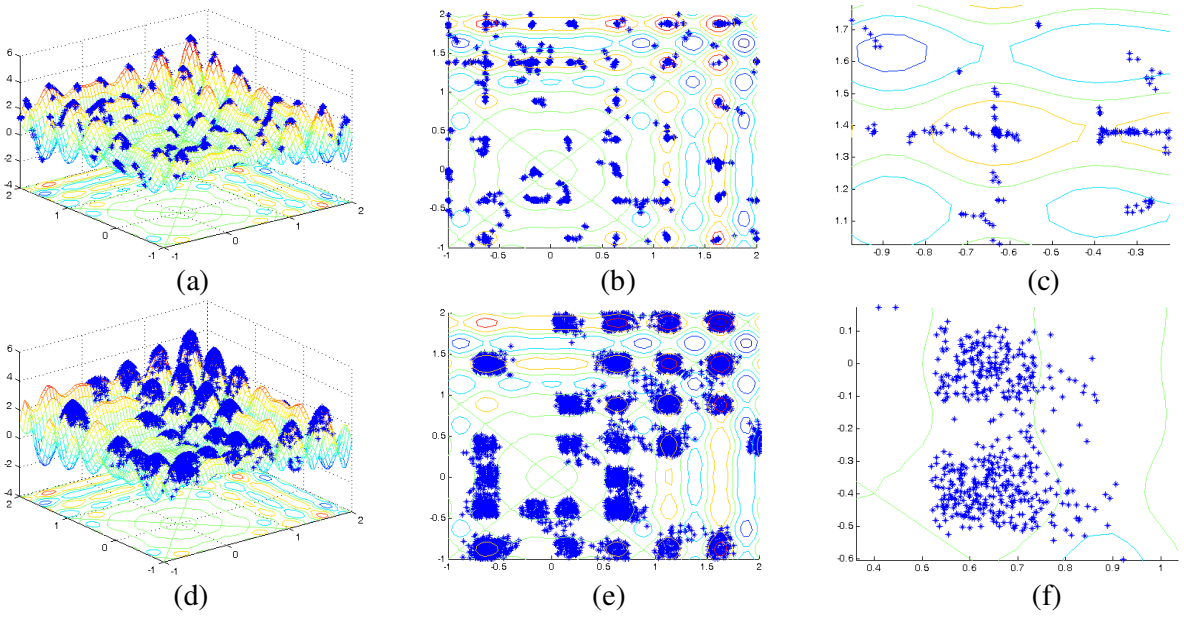

Figura 6.2- Processo de Hipermutação – Vetor tangente (a,b,c) e Utilizando indivíduos aleatórios (d,e,f)

6.5.2 Clusterização e Controle de Maturação em Otimização Contínua

Embora a adição do vetor tangente tenha contribuído para uma maior velocidade ao processo de convergência, o algoritmo não utilizou uma característica importante presente no sistema imunológico natural, conhecida como controle de maturação.

O controle de maturação é utilizado para aplicar a estratégia de agrupamentos no algoritmo de AIS. Esta especialidade permite encontrar simultaneamente vários ótimos locais.

Considere que em um dado sistema vários *Ab's* convirjam para ótimos locais. Após algumas iterações é possível gerar clusters representados por pontos que foram influenciados por um mesmo atrator. Uma vez que estes clusters estejam bem definidos, é possível aplicar o controle de maturação. Neste processo, em cada cluster são eliminados todos os *Ab`s*, exceto o

melhor. Esta estratégia pode reduzir consideravelmente a população inicial, além de melhorar a resposta computacional. Entretanto, é fundamental que os clusters estejam bem definidos.

A literatura mostra vários algoritmos para clusterizar conjuntos de dados. Neste trabalho foi utilizado o método da distância MaxMin (MMD) devido a duas grandes vantagens: a) o algoritmo estima automaticamente o número de clusters (o que é muito importante, já que não se conhece previamente o número de ótimos locais no sistema) e b) o algoritmo necessita somente de um parâmetro, o qual pode ser ajustado deterministicamente. O algoritmo segue os passos:

Algoritmo *MAXMIN*

0. Definir o parâmetro de distância mínimo (Dp);
1. Definir o Ab de maior afinidade como centro do cluster 1 e definir N_{cluster}=1;
2. Montar o vetor VD, onde VD*i* é a distância mínima entre Abi e todos os clusters centrais C*j* (j=1,Ncluster). Também armazena IDX*i* = j, onde j é o índice do cluster com a distância mínima para VDi;
3. Se o maior VD_I for maior que Dp então vá par 4, caso contrário, vá para 5;
4. Definir o Ab_I assossiado ao maior VD como sendo o centro de um cluster. Ir para 2;
5. Finalizar - os centro dos clusters estão armazenados em IDX.

Aplicando estes conceitos, a Figura 6.3 ilustra a evolução da população distribuída no espaço de solução, onde: Figura 6.3(a) mostra a população inicial, Figura 6.3 (b) e (c) exibem a primeira e a segunda geração, respectivamente, Figura 6.3 (d) e (e) mostram a população depois do controle de maturação e Figura 6.3 (f) uma projeção da equação. O número inicial de indivíduos para esta simulação foi de 100 anticorpos. Terminou com 82.

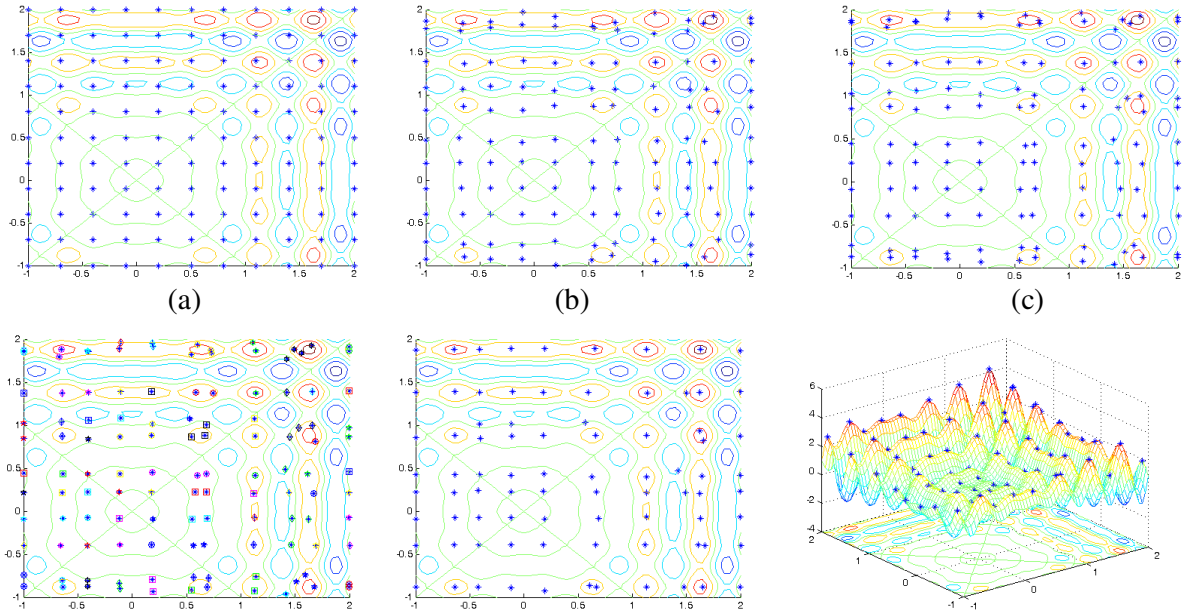

Figura 6.3- Processo de agrupamento, clusterização e controle de maturação.

De acordo com os resultados, algumas considerações são necessárias:

A primeira refere-se ao número de vezes que a clusterização ocorre. Como mostra a Figura 6.3, a partir da inicialização os anticorpos movem rapidamente para seus atratores. Realizar o controle de maturação pode produzir uma população escassa, mesmo nos primeiros estágios de evolução. Por esta razão, o processo é executado somente uma vez, na terceira geração.

A segunda está relacionada com o índice de distância, responsável pela determinação da construção do cluster. Se a população é gerada usando uma função de densidade com distribuição uniforme, a distância média entre os indivíduos é a medida do tamanho do espaço de solução pelo número de elementos.

Ainda que a idéia de aplicar a técnica de clusterização tenha sido implementada no Opt-AiNET e melhorada posteriormente, os conceitos adotados por estes algoritmos são diferentes dos propostos aqui. As principais diferenças são:

a) No algoritmo opt-AiNET o processo de clusterização acontece depois da primeira simulação, quando o critério de parada é encontrado, enquanto que o CGbAIS realiza este procedimento após poucas gerações.

b) O maior impacto do opt-AiNet em realizar esta clusterização é a supressão de memória (menos indivíduos armazenados no fim da simulação). Já no CGbAIS o propósito é o ganho de tempo computacional durante o processo de convergência

6.5.3 O Algoritmo CGBAIS

Analisando os progressos demonstrados até este ponto é possível concluir que a informação do Gradiente e o controle de maturação podem elevar consideravelmente o esforço computacional e a precisão aos métodos tradicionais baseados no AIS.

A proposta do algoritmo CGbAIS é unificar estes conceitos e melhorar os resultados. A Figura 6.4 apresenta o diagrama de blocos do método.

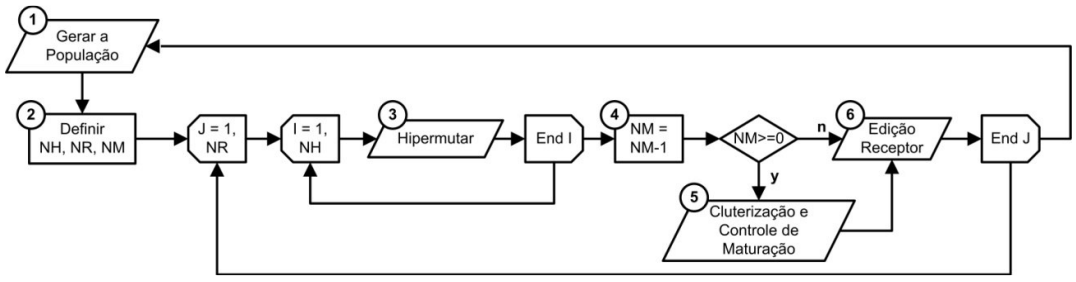

Figura 6.4- Fluxo do Algoritmo do CGbAIS

onde:

1. Gerar a população aleatoriamente.
2. Definir o número de hipermutaçoes (NH) realizadas antes do passo de clusterização, o número de vezes que o controle de maturação(NM) será executado, e finalmente o do receptor de edições (NR). É interessante frisar que com a utilização VT, a população tende rapidamente a se agrupar ao redor de um ponto estacionário, assim o NH pode ser muito menor que o NR.
3. Realizar o processo de hipermutação de acordo com a lista 1.
4. Diminuir NM para controlar o número de vezes que o controle de maturação ocorre. Conforme observações empíricas, este valor pode ser fixado em 1 ou 2. Após isto, a redução no número de indivíduos não é significativa.
5. Aplicar o Algoritmo (MMD) para clusterizar todos os indivíduos que convergem para um único atrator. Para cada cluster, eliminar todos os indivíduos, exceto o melhor.
6. Os melhores indivíduos dentro da população original são selecionados para a próxima geração. Os indivíduos restantes são substituídos por novos, gerados aleatoriamente. Este processo simula a edição de receptores e ajuda na busca por melhores soluções em áreas diferentes.

Como o algoritmo proposto consiste de duas estratégias (vetor gradiente e clusterização) é interessante identificar a relevância de cada estratégia separada e o algoritmo final. Quatro simulações foram realizadas. Para efeito de comparação a primeira foi realizada com o Clonalg (um dos principais algoritmos de AIS). Na segunda somente o vetor gradiente é considerado. A terceira somente o processo de clusterização, e quarta (CGbAIS) mostra todas as características (gradiente e cluster).

Conforme mostra a tabela 6.1, somente com o gradiente, *a taxa de sucesso* é melhor para todos os casos, o *tempo* é melhor em alguns. Analisando somente com clusterização, a *taxa de sucesso* é menor para todos os casos, o *tempo* também menor. O tempo foi melhor nesta

simulação já que foram eliminados indivíduos da população, mas como os indivíduos não foram guiados pelo gradiente a taxa de sucesso foi menor.

Com as duas técnicas juntas a *taxa de sucesso* é semelhante a do gradiente e o *tempo* semelhante ao cluster. A *taxa de sucesso* foi melhor para todos os casos e o *tempo* foi melhor para quase todos (neste caso existe uma diferença devido a população ser muito pequena, assim o processo de clusterizacao não eliminou muitos indivíduos).

Com o algoritmo proposto quanto mais indivíduos melhor o tempo computacional comparado ao CLONALG.

Tabela 6-1 Resultado das simulações do CGBAIS e variações

			CLONALG		GRADIENT ON		CLUSTER ON		CGbAIS	
Pop	Gen	Pop.Fixa	Taxa Suce%	T. Med(s)	Taxa Suce%	T. Med(s)	Taxa Suce%	T. Med(s)	Taxa Suce%	T. Med(s)
20	10	15	42	0.09	59	0.11	39	0.08	61	0.09
40	10	20	88	0.25	91	0.2	83	0.17	93	0.1
60	10	30	97	0.47	100	0.29	91	0.23	100	0.1

6.5.4 Problemas de Otimização Discreta

Para adaptar os princípios do CGbAIS na otimização discreta, são necessárias algumas observações:

a) Um anticorpo representa um caminho parcial em uma árvore de busca. O nível começa no nível 1, conhecendo apenas um nó, e de acordo com a evolução, novos nós são adicionados. Isto significa que a população inicial pode pesquisar em todo o espaço de solução, e durante o processo de evolução são direcionadas a áreas interessantes.

b) Em cenários discretos, qualquer mudança nas variáveis de controle (neste caso representado por um caminho na árvore) pode levar o sistema a um estado operacional completamente diferente, tornando incorreta a idéia de pequenas mudanças no estado corrente.

A estratégia adotada durante a hipermutação foi de manter os *nlocalbest* clones, ao invés de apenas um. Assim o algoritmo permite que um determinado anticorpo que não foi bem classificado inicialmente, possa evoluir e tornar-se melhor. Este conceito é o mesmo de outros algoritmos bem sucedidos como o *Simulate Annealing* ou busca *Tabu*. Para evitar uma explosão combinatória, a cada iteração, todos os clones são comparados e somente os melhores *nGlobalBest* são mantidos pelo controle de maturação.

c) Em qualquer aplicação, a condição de um elemento de um dado cluster deve seguir algum indicador de similaridade. No caso da otimização contínua, a população inicial de anticorpos evolui e os clusters são encontrados utilizando o algoritmo de distância MAXMIN. Na otimização combinatória, o trajeto na pesquisa da árvore contém informações mais valiosas que o resultado final, o que indica que o caminho que o elemento evoluiu pode ser usado como medida de similaridade. Desta forma, se um elemento tem os mesmos n-primeiros nós que outro, eles podem ser clusterizados junto no nível-n.

d) O processo de hipermutação ocorre em cada anticorpo onde os clones *nLocalBest* são mantidos. Como declarado anteriormente, o controle de maturação compara os *nLocalBest* de todos *Ab's* e somente mantêm os *nGlobalBest*. Neste processo todos os clones de um dado anticorpo podem ser eliminados, o que representa a eliminação de ramificações na árvore de busca. Desta forma, uma boa estratégia é definir o valor *nLocalBest* menor que o de *nGlobalBest* (o que evitaria uma eliminação prematura de possíveis caminhos de busca que não são bem classificados em um primeiro momento, mas são promissores). Uma boa estratégia de ajuste permite explorar um bom número de caminhos sem eliminar muitos filhos.

e) Os clones são responsáveis por analisar um dado nó. A população expande os nós dos seus pais e as informações numéricas obtidas por este processo indicam a probabilidade de encontrar a melhor solução em relação a um determinado caminho. Porém, o sucesso da idéia depende da exatidão da informação numérica, que apresenta dois grandes problemas:

O primeiro é como evoluir corretamente em cada ramo. Se somente o valor mínimo de um clone for considerado, significa que todos os outros são negligenciados e informações

valiosas podem ser perdidas. Para evitar o problema, a função de avaliação utilizada no algoritmo é mostrada a seguir:

$$VT = v_i \times \Phi\{v_1, \ldots, v_{nLocalbest}\} \tag{6.5}$$

A função pega o valor médio de Φ, fornecido pelo *nLocalbest* individual de cada ramo e o multiplica por cada clone individual de valor v_i.

O segundo problema é que, se na árvore de buscar tiver dois ou mais ramos com taxas de sucesso parecidas, a probabilidade de não encontrar a melhor no final da simulação vai depender da proximidade destes e do número de clones utilizados para obter as informações necessárias. Neste caso, a solução final pode ser próxima ao melhor global e melhorada facilmente por uma busca local (hipermutação).

A Figura 6.5 ilustra o processo de como a busca na árvore é realizada. O principal objetivo é expandir um nível da árvore a cada geração. O processo começa no estado inicial 0 e deste, uma população de tamanho n é gerada, expandindo o primeiro nível da árvore. O processo de clonagem é responsável por definir qual ramo é mais interessante para a próxima pesquisa. Para cada indivíduo, que representa um caminho parcial da árvore, é selecionado um caminho aleatoriamente e avaliado de acordo com a equação 6.5. Os *nLocalBest* clones de cada anticorpo serão selecionados e armazenados em uma lista comum.

Uma observação importante é que os clones formam caminhos completos sob a árvore, e podem ter tamanhos diferentes. Os *nGlobalBest* clones são selecionados, gerando anticorpos com um nível a mais que o indivíduo original, representado neste exemplo pelo nível 2. Clusterizar estes indivíduos significa encontrar os indivíduos repetidos e eliminá-los, com exceção de um. Na Figura 6.5, o clone [0-1-a1-b1-c1]e o [0-1-a1-b1-d1-f1] tem os mesmos dois primeiros nós, [0-1], representando o nível-2 de similaridade, logo eles podem ser clusterizados. Note que os anticorpos da população final estão apenas um nível a mais de profundidade na árvore que os iniciais, o que significa que crescem um nó a cada geração.

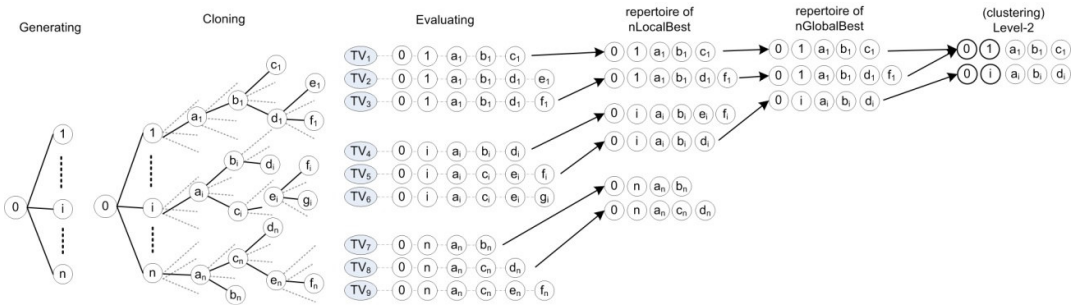

Figura 6.5- Otimização combinatória utilizando o algoritmo CGbAIS

6.5.5 Problemas não-Lineares com Restrições – O Algoritmo α-CGbAIS

Como já mencionado, um problema de otimização não-linear pode ser matematicamente descrito como:

$$\begin{aligned} &\textit{Minimize} && \mathrm{f(x)} \\ &\textit{S.a.} && \mathrm{g(x)} = 0 \\ & && h(x) \le h_{\max} \end{aligned} \qquad (6.6)$$

Onde, x é um vetor de entrada (n*x*x1) ou variáveis de controle, *f(.)* é uma função escalar representando o objetivo a ser otimizado, *g(.)* é conjunto de funções representando as restrições, e *hmax* é um vetor (n_hx 1) de restrições associados com os limites das funções de restrições de desigualdade.

Existem vários caminhos para resolver a equação 6.6. Eles podem ser divididos em categorias conforme o tratamento das restrições. As principais são descritas a seguir:

- "*Death penalty*": neste método um ponto infactível é descartado ou reparado. É difícil aplicar quando o espaço de solução é pequeno ou o número de variáveis é grande.

- "*Penalty functions*": este combina as restrições com a função objetivo conforme as penalidades. É utilizado também em métodos numéricos que expandem a função lagrangeana com penalidade de barreira. O principal problema é saber qual peso a penalidade deve ter em relação ao objetivo. Se o coeficiente de penalidade é grande, a solução factível pode ser encontrada, entretanto a função objetivo pode não ser otimizada corretamente, gerando uma solução possível, porém ruim.
- "*Relaxation of Constraints*": as restrições e a função objetivo são otimizadas juntas adotando a ordenação lexográfica com relaxamento das restrições. Neste método todas as restrições que têm o limite violado são ignoradas. Este limite se reduz a zero durante a simulação.

Como demonstrado em [40] esta última técnica é mais eficaz para diversos problemas. Os autores usam um nível de satisfação (μ_α (x)) que indica o quanto um ponto x satisfaz as restrições relaxadas em α. Assim, se um determinado ponto x satisfaz todas as restrições, o valor de μ(x) será 1. Se não satisfaz as restrições, mas a violação está sob um índice positivo definido, μ(x) estará entre 0 e 1. Caso contrário, o valor é 0.

Para um melhor entendimento deste método, chamado *α-constrained,* a figura 6.6 exibe um exemplo. Um ótimo global sem restrições é representado por "*ugo",* o sistema restrito por "*C*" *(α=1),*com ótimo global restrito "*cgo*". Devido a forma da restrição apresentada, é muito difícil achar este ponto ótimo.

Usando o método baseado no *α-constrained , "*C" é relaxado através de um *α* (o valor 0 indica totalmente relaxado e 1 não relaxado). Este valor vai indicar o quanto está sendo relaxada a restrição em questão.

Para o exemplo*, α* assume o valor *0,1,* levando a uma situação onde *ugo=cgo*.Neste caso o ótimo é encontrado. Deste ponto, *α* é lentamente incrementado e o *cgo* seguirá a restrição ate *α=1*.

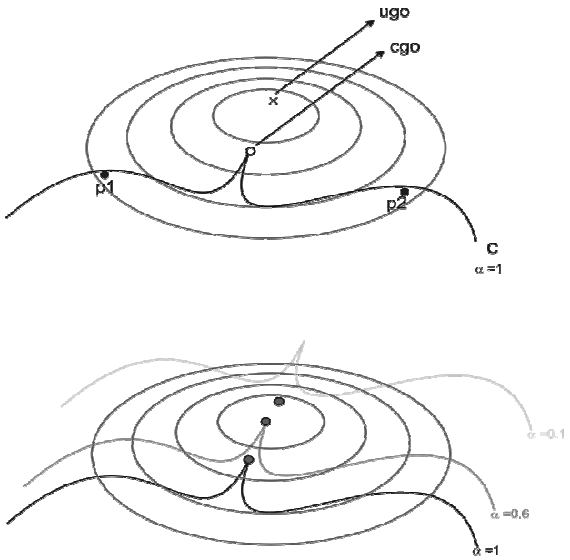

Figura 6.6- α-Constrained

Neste trabalho, o nível de satisfação é calculado de acordo com a função de penalidade Φ.

$$\Phi(x) = \sum_i \max\{0, |g_i(x)|\} + \sum_j \max\{0, |h_j(x) - h_{\max}|\} \tag{6.7}$$

$$\mu(x) = \max\left\{0, 1 - \frac{\Phi(x)}{B}\right\} \tag{6.8}$$

$$B = \frac{\sum_{i=1}^{npop} \Phi_i}{npop} \tag{6.9}$$

Onde b é um número obtido pela população inicial com tamanho da população.

Com o nível de satisfação é possível determinar a comparação lexicográfica. μ(x) é considerado antes de f(x), já que a viabilidade é mais importante que a realização da função objetivo. Para efeito de simplificação, o vetor (f(x), μ(x)) será definido como $F_\alpha(x)$. A seguir é exibido o teste para identificar se $F_\alpha(x_1) \leq \alpha\ F_\alpha(x_2)$.

$$\begin{array}{ll} \textit{If} & \text{Test} \\ \mu_1, \mu_2 \geq \alpha, & f_1 \leq f_2, \\ \textit{caso contrário}, & \mu_1 \geq \mu_2. \end{array} \tag{6.10}$$

Com estes conceitos o problema é definido como:

$$\begin{array}{l} \textit{Minimize} \quad F_\alpha(x) \\ \text{S.a} \quad \mu(x) = 1 \end{array} \tag{6.11}$$

No início da simulação, α é relaxado, possibilitando uma busca expandida no espaço de solução. Conforme a simulação evolui, α tende a 1. Isto assegura que as soluções factíveis sejam mais bem classificadas.

Como exemplo, suponha dois pontos operacionais, com {f, Φ} com op_1 = (1.2, 2) e op_2= (1.1, 2). O objetivo é minimizar f com B=10. Para este valor de B, o nível de satisfação μ_1 de $op_1 = 1$ e μ_2 *de* $op_2 = 0{,}8$. Para definir o melhor ponto consideramos o α -nível de satisfação. Se α=0,5, μ_1 *e* μ_2 satisfazem o relaxamento, neste caso escolho o ponto com função objetivo menor, no caso op_2. Porém, se α=0,9 somente op_1 satisfaz o relaxamento, logo escolho op_1 indiferentemente do valor da função objetivo, já que encontrar a factibilidade é mais importante que reduzir a função objetivo.

Entretanto, como a técnica proposta por este trabalho é baseada no vetor tangente, se o ponto ótimo estiver perto das restrições, o sistema pode apresentar alguns problemas de convergência. Nestes casos, a técnica do vetor gradiente fornece pequenas melhoras em direção ao ótimo. Para evitar tais situações, dentre outras desvantagens, algumas modificações foram realizadas e descritas a seguir:

O problema principal na otimização restrita para os algoritmos baseados em população está nas restrições de igualdade. Se o problema é não-linear e de grande escala (com milhares de variáveis) a situação é pior. Em aplicações como [11,14,41] é recomendado resolver primeiro as restrições de igualdade utilizando métodos numéricos e só então evoluir a população.

Neste trabalho, as restrições de igualdade serão solucionadas pelo método de busca apresentado em [42]. Para unir os conceitos aos problemas de otimização que usam algoritmos baseados em população é necessário determinar as variáveis de entrada e saída. Os métodos numéricos adotados aqui usam as variáveis de entrada (x_i) como ponto de partida. A solução é encontrada pela função SolveG, responsável por gerar um novo conjunto de variáveis definidas como de saída. As variáveis de saída (x_o) serão usadas para avaliar a função objetivo e fornecer o valor final $f(x_o)$. Isto significa que as variáveis usadas com entrada para um indivíduo não são as mesmas usadas para avaliar a função objetivo. Consequentemente várias entradas irão convergir para a mesma direção das variáveis de saída, onde cada conjunto é uma solução de $g(x_o) = 0$ e um subespaço no problema de otimização.

A segunda modificação do método α-constrained está na adição da barreira logarítmica e variáveis de folga (s). O novo problema será descrito como:

$$\begin{aligned} &\textit{Minimize} && \mathrm{F}_{\alpha}(\mathrm{x}_0) - \sigma_{\mathrm{k}} \sum \ln(s) \\ &\text{S.a} && \mu(\mathrm{x}_0) = 1 \\ &\textit{onde} && \mathrm{s} = \max\{\delta, \mathrm{h}_{\max} - h(x_0)\} \\ & && x_0 = SolveG(x_i). \end{aligned} \tag{6.12}$$

O parâmetro δ é um número pequeno (i.e. 0.0001) para evitar que a barreira logarítmica seja infinita, σ_{k} é um parâmetro da barreira que varia entre 0 e 1 durante a simulação. A equação abaixo controla este valor.

$$\sigma_k = 1 - \alpha \tag{6.13}$$

Se o problema não tiver restrições de igualdade, xo = xi.

Esta estratégia combinada permite a busca por áreas melhores para relaxar as restrições. Também aumenta o processo de convergência e evita limites formados pelo espaço de solução.

A terceira modificação é com relação ao modo que o vetor tangente é computado. Acoplando o conceito ao método α-constrained, temos o VT como:

$$\begin{aligned} &if\,(\mu_1, \mu_2 \geq \alpha), && \mathrm{TV} = (\mathrm{f}_2 - f_1)/(\mathrm{x}_2 - x_1), \\ &caso\,contrário, && \mathrm{TV} = (\mu_2 - \mu_1)/(\mathrm{x}_2 - x_1). \end{aligned} \tag{6.14}$$

A quarta modificação está relacionada às características não elitistas do AIS. Em casos de otimização onde as restrições são tratadas como incertezas, é mais vantajoso ter soluções múltiplas ao invés de somente uma. Nestes casos, um conjunto de soluções de ótimos locais é analisado, e de acordo com as incertezas o mais adequado é escolhido.

Na edição de receptores, os indivíduos são serão substituídos aleatoriamente, estes serão calculados segundo as equações 6.15 ou 6.16.

$$Ab_k = a \times Ab_j + (1 - a) \times Ab_k \tag{6.15}$$

$$Ab_k = Ab_k + (b - 2b \times rand) \tag{6.16}$$

Onde, k é o anticorpo que passa pela edição de receptor, j é um dos *j-melhores* anticorpos, "a" é um número real variando de 0 a 2, "b" é um número real positivo.

O resultado final ainda é um algoritmo não-elitista que irá convergir para os *j-melhores* anticorpos. No entanto, mais indivíduos são atraídos para estas regiões, aumentando a possibilidade de melhorar o processo de convergência. É importante notar que 6.16 tem o objetivo de evitar uma rápida convergência e também melhorar a exploração da vizinhança pela busca de áreas mais promissoras. Para melhorar a diversificação o valor de "b", comparado ao

limite da variável, não deve ser alto. O algoritmo irá definir a cada iteração se a equação 6.15, 6.16 , ou ambas serão utilizadas.

A quinta modificação é uma adaptação de [43]. A proposta original é de classificar probabilisticamente os indivíduos não factíveis na população. Este critério força a diversificação da população produzindo melhores resultados. A proposta deste trabalho é usar esta idéia e comparar o individuo original com o clone. Se o clone for melhor, a substituição é feita, mas se for pior, a substituição também pode ser feita com uma probabilidade *Pm*, caso a diferença entre seus níveis de satisfação estejam abaixo de um nível de degradação *adl* como apresentada a seguir em 6.17.

$$\begin{aligned} &\text{if } (\mathrm{F}_{\alpha}(\mathrm{cAb}) < \mathrm{F}_{\alpha}(\mathrm{Ab})) \\ &\quad \text{or } (\text{ rand} < Pm \; and \; \left| \mu_{\alpha}(\mathrm{cAb}) - \mu_{\alpha}(\mathrm{Ab}) \right| < adl \;) \\ &\qquad \text{then } \{\, \mathrm{Ab} = \mathrm{cAb} \,\} \end{aligned} \tag{6.17}$$

Embora este conceito esteja presente em [43], o objetivo e o impacto no processo de convergência são diferentes. O processo é parecido com o "*simulating annealing*", onde o algoritmo permite aceitar vizinhos piores com uma certa probabilidade, para posterior melhora.

Finalmente, a última modificação é uma nova definição de como evoluir a população. Alguns métodos, como *Swarm* [14,44], e *Differential Evolution* [45,46] mostram um forte relacionamento entre a população e o processo de convergência. Em outras palavras, todos os indivíduos da população são usados, e a cada geração compartilham informações para direcionar a busca. Já o α-CGBAIS, conta com o vetor gradiente, e a linha de busca é produzida pelos melhores e piores indivíduos. Desta forma, é possível, a cada geração, evoluir somente os melhores e piores indivíduos, economizando um tempo computacional considerável.

Alguns passos no CGBAIS foram adaptados para o α-CGBAIS, como segue:

1. Escolher os parâmetros {*npop, nbest,nworst,MaxInt,clusterAt,Pm,adl,a,b*}.

2. Gerar a população aleatoriamente;

Para *nint*=0 até *MaxInt*

3. Avaliar cada Ab usando FPO tradicional – este passo assegura g(x)=0;
4. Calcular para cada Ab_j, $F_\alpha(Ab_j) = \{f(x_j) - \rho\sigma_k \sum \ln(s_j), \mu(x_j)\}$
5. Ordenar a população de acordo com (6.10);
6. Selecionar os indivíduos *nbest* e usar (6.4 e 6.14) para gerar seus clones.
7. Substituir os clones se satisfazem (6.17);
8. Selecionar os piores indivíduos para o Receptor de Edição.
    a. Gerar novos indivíduos de acordo com(6.15 / 6.16)
    b. Substituir se atender (6.17);
9. Se *nint*=*ClusterAt* rodar o algoritmo *MMD*

Onde, *MaxInt* é o número máximo de iterações e *ClusterAt* é a iteração onde o processo de clusterização ocorre.

6.6 Validação e Testes do Algoritmo CGbAIS

Para validar o algoritmo CGbAIS foram realizados três testes. Serão considerados o número de avaliações e a taxa de sucesso.

6.6.1 Otimização sem Restrições

No primeiro caso, um conjunto de problemas sem restrições utilizam as equações 6.2,6.18 (onde i representa a unidade imaginária) e 6.19, identificadas como funções 1,2 e 3 respectivamente. O opt-AINet foi avaliado por estas funções, como pode ser visto pelas referências [39,47]. O resultado apresentado por este algoritmo será comparado com o proposto (CGbAIS).

$$f(z)=\frac{1}{1+\left|z^6-1\right|},\ z=x+yi \tag{6.18}$$

$$f(x,y)=0.5+\left\{\frac{\sin^2\left(\sqrt{x^2+y^2}\right)-0.5}{1+0.001\times(x^2+y^2)}\right\}. \tag{6.19}$$

As Figuras 6.7 e 6.8 exibem as funções 2 e 3.

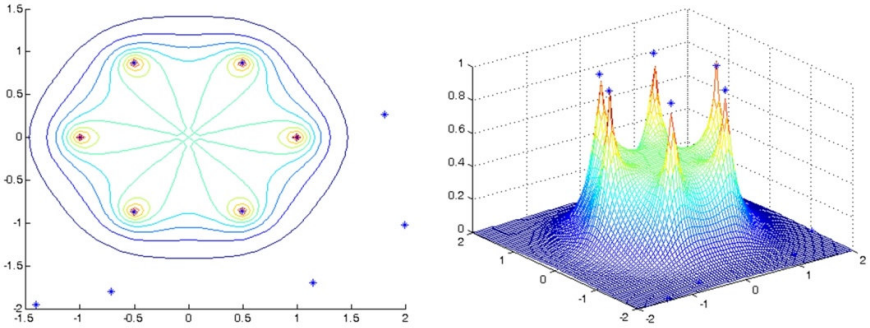

Figura 6.7- Função Roots

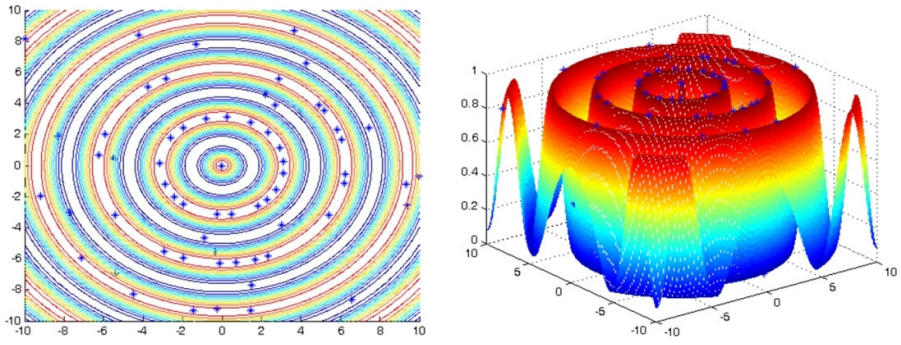

Figura 6.8- Função Teste Shatter

A Tabela 6-2 apresenta os resultados obtidos pelos CGbAIS e opt-aiNET após 20 iterações. Os algoritmos serão comparados segundo os parâmetros: tamanho da população inicial (*Spop*), tamanho da população final (*Fpop*), número de avaliações de f(x) (*Neval*), melhor anticorpo (*Bab*), valor médio dos 50% melhores anticorpos (50%*BAb*), taxa de sucesso representando a porcentagem que o ótimo global foi encontrado (*sr*), o número de gerações até a convergência (itG) e o índice fornecido pelo número de avaliações para o opt-aiNet e CGbAIS. Os parâmetros utilizados no opt-aiNET são os mesmos apresentados pela referência [39]. As figuras 6.6 e 6.7 mostram as funções 2 e 3. Observe que a função 2 tem seis cumes, logo a taxa de sucesso computacional é o valor médio do total de cumes pelo ótimo global encontrado.

Tabela 6-2 Resultado das Simulações do CGBAIS e do Opt-aiNet

			CGbAIS						opt-aiNet						Neval Rating
#	Func	Spop	Fpop	Neval	Bab	50%BAb	sr	itG	Fpop	Neval	Bab	50%BAb	sr	ItGlobal	opt/CGb
1		20	6.750	760.65	4.004	3.548	0.600	26.500	50.850	48,421.50	3.402	2.211	0.250	136.150	63.658
2		40	8.100	1,110.00	4.129	3.586	0.750	26.000	66.850	68,837.50	3.709	2.243	0.300	136.650	62.016
3	1	60	8.800	1,301.85	4.179	3.660	0.850	23.450	72.900	79,228.50	3.699	2.221	0.350	128.450	60.858
4		80	9.250	1,556.55	4.178	3.607	0.850	22.750	95.550	119,783.50	3.804	2.290	0.350	142.850	76.954
5		100	9.950	1,750.35	4.254	3.628	1.000	21.850	92.600	119,606.50	3.943	2.260	0.370	138.650	68.333
6		20	6.800	764.55	0.999	0.989	1.000	28.800	33.650	73,534.00	0.861	0.481	0.000	283.200	96.179
7		40	8.150	1,048.65	1.000	0.992	1.000	22.500	52.350	143,633.00	0.919	0.514	0.000	330.550	136.969
8	2	60	9.050	1,316.40	0.999	0.993	1.000	23.150	60.900	198,894.50	0.944	0.497	0.050	346.900	151.090
9		80	8.900	1,509.60	1.000	0.992	1.000	21.000	67.350	230,760.00	0.952	0.489	0.100	324.850	152.862
10		100	9.300	1,709.85	0.999	0.991	1.000	21.450	74.150	287,926.00	0.952	0.486	0.110	354.100	168.393
11		20	16.300	4,434.45	0.996	0.989	0.250	102.850	43.350	134,008.00	0.994	0.953	0.000	454.600	30.220
12		40	26.500	7,486.50	0.995	0.989	0.250	94.850	82.300	265,826.00	0.995	0.959	0.000	475.950	35.507
13	3	60	34.850	10,448.25	0.997	0.990	0.350	93.000	113.400	392,023.50	0.995	0.955	0.050	481.750	37.520
14		80	41.150	11,695.95	0.997	0.990	0.500	82.050	164.050	539,858.00	0.995	0.954	0.000	488.200	46.158
15		100	46.950	13,273.80	0.998	0.990	0.600	78.050	179.650	644,405.00	0.995	0.957	0.050	493.950	48.547

De acordo com os resultados fornecidos é possível concluir que o CGbAIS é mais confiável e eficiente. Ele converge com um número menor de iterações e com taxa maior de sucesso para todos os casos. Como exemplo, observe que nos casos 6, 7, 11, 12 e14 o opt-aiNET não encontrou o ótimo global, enquanto que o CGbAIS teve uma taxa de sucesso de 0.88, 0.88, 0.25, 0.25 e 0.50. Na população final com o CGbAIS, o número de gerações necessárias para encontrar o ótimo global foi menor para todos os casos simulados. O melhor anticorpo, o índice 50%Bab, e a taxa de sucesso obtiveram maior ganho utilizando o algoritmo proposto que com o opt-aiNET. Em relação ao número de avaliações é possível observar que o CBbAIS utilizou de 30 a 168 vezes menos avaliações que o opt-aiNET.

A figura 6.9 apresenta os resultados para uma população inicial de 100 indivíduos. É descrita a diferença de desempenho dos índices Bab, 50%Bab e nPop. Este último é calculado de acordo com o tamanho da população corrente pelo número de indivíduos obtidos durante as gerações.

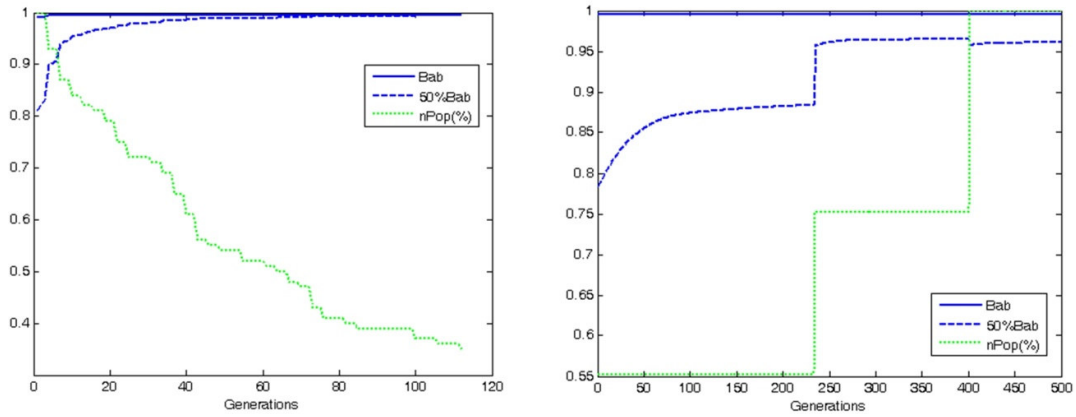

Figura 6.9- Gráfico Comparativo entre os Algoritmos Opt_AiNet e CGbAIS

É possível perceber que o CBbAIS, 6.8(a), diminui assintoticamente o tamanho da população sem perder precisão, no opt-aiNET, 6.8(b), o tamanho da população aumenta sem conseguir a mesma precisão. Neste exemplo, o opt-aiNET terminou a população com 181 indivíduos após 500 gerações e, mesmo assim, o ótimo global não foi encontrado. Já o CGbAIS terminou com 46 anticorpos e encontrou o ótimo global com 118 gerações.

A seguir as principais razões para explicar a diferença de desempenho:

Em primeiro lugar, utilizar VT é mais vantajoso por ser mais confiável e eficaz que métodos aleatórios. Esta estratégia direciona o indivíduo rapidamente a um ponto estacionário. A segunda razão é que utilizando VT é possível determinar se um dado anticorpo alcançou um ponto estacionário. Caso alcance, o mesmo fica bloqueado para novas avaliações, poupando tempo computacional. Terceiro ponto: a clusterização e eliminação de indivíduos têm abordagens e significados diferentes para os algoritmos. Enquanto no CGbAIs é representado pelo controle de maturação, no qual um indivíduo não se desenvolve se existe um similar

melhor que ele, no opt-aiNET este passo é representado pela supressão de memória, onde, somente após a geração completa os indivíduos similares são eliminados. A principal diferença do algoritmo proposto está na eliminação prematura de indivíduos redundantes, o que proporciona uma economia considerável no esforço computacional. A última razão é que com a utilização destas técnicas é possível diminuir a população e ainda reduzir o número de avaliações.

O opt-aiNET aumenta a população para pesquisar no espaço de solução. Conseqüentemente, muitas avaliações são necessárias, elevando o esforço computacional sem aumentar a precisão.

6.6.2 Otimização Combinatória

O segundo caso utiliza o problema do caixeiro viajante (PCV) para avaliar a eficácia do algoritmo CGbAIS em problemas de otimização combinatória. O PCV é um problema clássico de otimização e certamente um dos mais estudados. Consiste em, dada uma lista de cidades encontrar uma rota que permita visitar todas, começando e terminando numa mesma cidade de modo a tornar mínimo o percurso total.

Vários métodos heurísticos, como 2-opt [48] , 3-opt [49] além de outros, resolvem este problema de forma eficaz. Porém, em grandes problemas métodos baseados em heurísticas independentes [50,51], como os algoritmos genéticos [52], não apresentam bom desempenho.

A proposta de testar o PCV usando CGbAIS não é de competir com os algoritmos com domínios dedicados, que são bem difíceis e específicos ou até mesmo impossíveis de serem empregados em alguns problemas combinatórios. O principal objetivo é demonstrar a eficácia do algoritmo para problemas em geral.

Considerações necessárias para o entendimento de como o CGbAIS foi adaptado ao PCV:

a. Cada anticorpo é um trajeto ao longo das cidades, começa com a cidade inicial e final. Todas as outras cidades são espaços vazios, e a cada nova geração uma cidade é adicionada.

b. Um clone é um caminho parcial aleatório, que representa os espaços vazios do anticorpo.

c. O vetor tangente de um dado anticorpo é avaliado de acordo com a equação 6.5, que pode ser entendida como o produto do valor médio dos nbest clones por seu valor mínimo.

d. O índice de similaridade utilizado para clusterizar os indivíduos é determinado por níveis, onde um n-nível similar indica que dois anticorpos possuem exatamente as mesmas n cidades anteriores.

Um dos problemas encontrados é que, se duas cidades forem próximas, apesar de probabilidades semelhantes, podem apresentar soluções diferentes. Neste caso a solução final pode ser próxima ao ótimo global e então melhorada através de uma busca local (hipermutação).

A Figura 6.10 ilustra uma situação para o problema com 20 cidades, algumas bem próximas a outras. Isto, juntamente com o fato de que poucos clones foram utilizados, definiu a primeira solução (Figura 6.9a) como não muito longe do ótimo global.

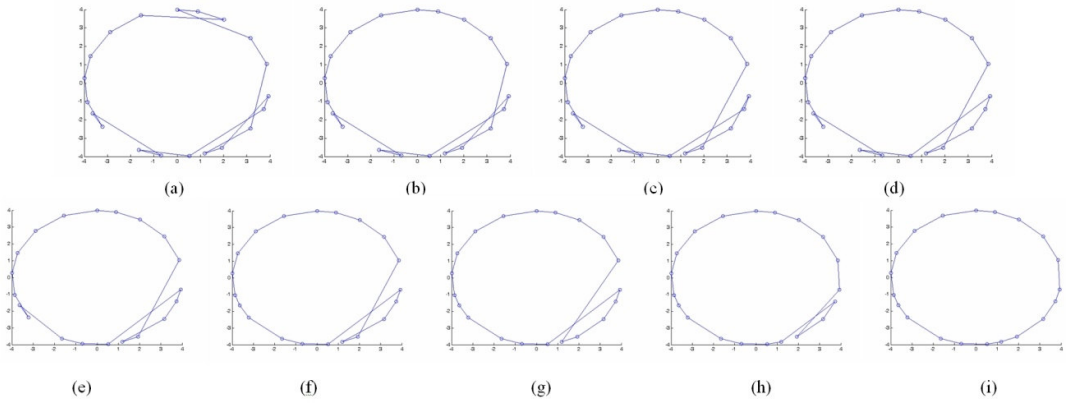

Figura 6.10- Exemplo do Processo de Hipermutação

Para melhorar a resposta final, a referência [50] mostra um processo de hipermutação, onde uma seqüência de n cidades {C1...Cn} permuta Ci com Cj.

O algoritmo apresentado melhora o original por permitir quantas permutações forem necessárias para encontrar um ponto estacionário (nenhuma solução melhor na região.)

As cidades 1 e 2 foram selecionadas para alteração. Após cada processo de hipermutação, as inconsistências encontradas durante a primeira etapa são eliminadas. A distância diminuiu, o fator de precisão é melhorado e, após 8 iterações, o sistema encontra o ótimo global, 6.9 (i).

A seguir, três conjuntos de simulações medem o desempenho do algoritmo. Foram utilizados 20 casos com 30 cidades para o PCV. Os números de *nLocalBest* e *nGlobalBest* serão os mesmos e as respostas correspondem ao total de 100 simulações. A cada simulação, um novo cenário será criado.

O primeiro conjunto caracteriza-se por casos onde o número de clones e os anticorpos *nbest* variam de 10 a 50. O segundo é composto de casos onde o número de clones varia de 60 a 100, e os anticorpos *nbest* são definidos como 40 e 60 para todos os casos, com o mesmo tamanho da população. O terceiro conjunto é para testar somente a hipermutação final como solução do algoritmo. Logo, o número de clones e anticorpos *nbest* não são aplicados. A idéia é gerar vários casos com parâmetros diferentes e analisar a resposta de acordo com o número de avaliações direcionadas a cada um.

A tabela 6.3 mostra os resultados, onde *Ncl* é o número de clones de cada anticorpo, *nbest* é o número dos melhores clones: local e global, Sr é a taxa de sucesso final (após o processo de hipermutação), também representa a taxa de casos que alcançaram o ótimo global. *Acc* é o valor médio final de precisão dado pela equação 6.20.

Tabela 6-3 Resultados do CGbAIS no TSP

Case	*Ncl*	*nbest*	*sr*	*Acc*	*stdAcc*	*Eval (x1000)*
1	10	10	0.51	0.79	0.22	6.06
2	20	20	0.76	0.90	0.17	11.85
3	30	30	0.93	0.97	0.11	21.70
4	40	40	0.94	0.98	0.09	35.13
5	50	50	0.96	0.98	0.08	52.34
6	60	40	0.93	0.97	0.10	49.74
7		60	0.99	1.00	0.04	72.67
8	70	40	0.96	0.98	0.08	57.20
9		60	0.98	0.99	0.06	83.18
10		70	0.99	1.00	0.03	96.44
11	80	40	0.96	0.98	0.08	64.53
12		60	0.99	1.00	0.04	94.35
13		80	0.99	1.00	0.04	123.74
14	90	40	0.95	0.98	0.09	71.78
15		60	0.99	1.00	0.04	104.88
16		90	1.00	1.00	0.00	154.68
17	100	40	0.98	0.99	0.05	79.60
18		60	0.99	1.00	0.04	115.24
19		100	1.00	1.00	0.00	188.69
20	:	:	0.11	0.57	0.16	4.58

$$\Pr ecisão = \frac{MelhorSolução}{SoluçãoCorrente} \tag{6.20}$$

A equação avalia a Melhor Solução (ex: ótimo global) pela Solução Corrente. O parâmetro *stdAcc* é o valor do desvio padrão dos casos precisão, *Eval* é o valor médio das avaliações.

Inicialmente será analisada a terceira simulação, o caso 20 (onde somente a hipermutação final foi testada). Os resultados demonstram que o uso somente desta metodologia não fornece bons resultados. Foi o pior caso simulado, somente 11% de taxa de sucesso e 0.57 de precisão. Este, não tem nenhum parâmetro a ser alterado e o número de cidades é a única variável.

Note que, a inclusão da hipermutação ao processo leva a resultados melhores e mais confiáveis. Casos com mais de 80.000 avaliações conseguiram taxa de 99% de sucesso e precisão maior que 0.99. Isto também mostra que valores baixos de desvio padrão implicam em resultados sem grandes variações. Comparando estes resultados com o algoritmo fornecido pela referência [21], casos com aproximadamente 140.000 avaliações tiveram taxa de sucesso de 71% com precisão de 0.92.

Aumentando o número de cidades, a diferença é ainda maior. Em um cenário com 50 cidades, o algoritmo proposto tem taxa de sucesso de 60% contra 10% obtidos pelo CLONALG. A simulação contou com 410.000 avaliações e a hipermutação final foi considerada em ambos os algoritmos (visando uma comparação justa).

Outros resultados também são exibidos pela Tabela 6.2 e demonstram que o método além de confiável, não é sensível ao número de clones e população *nbest*.

O fator mais importante para gerar bons resultados está no número de avaliações. Na tabela os casos são ordenados conforme este número. Analisando os resultados, percebe-se que a taxa de sucesso (*sr*) e o valor médio de precisão (*Acc*) aumentam quando o número de avaliações aumentam.

6.7 Validação e Testes do Algoritmo α-CGbAIS (otimização com restrições)

O último caso analisa a eficiência do algoritmo para problemas de otimização restrita. Será utilizado o α-*Constrained* CGbAIS, mostrado anteriormente, e os problemas G1, G7, G9, G10, retirados de [18] e G11 [40]. Serão realizadas comparações com o modelo co-evolucionário do lagrangeano aumentado (CAL) e com o método do α-*Constrained* (αC) apresentados pelas mesmas referências.

A Tabela 6.4 mostra os resultados após 30 iterações. A população iniciou com 200 indivíduos, terminou com 6. Foram utilizados dois ciclos de clusterização durante o processo de convergência.

Tabela 6-4- Comparaçao entre os Métodos de Lagrengeano Aumentado, Alfa-Simplex e Afla-CGbAIS

Problem	G1		G7		G9		G10		G11	
Optimal	-15.000000		24.306209		680.630057		7049.248000		0.750000	
Result Type	*best*	*worst*	*best*	*worst*	*best*	*worst*	*best*	*worst*	*best*	*worst*
augmented Lagrangian	-15.000000	-15.000000	24.306000	24.308000	680.630000	680.630000	7050.169000	7136.600000	n.a.	n.a.
alfa simplex	-15.000000	-15.000000	24.306210	24.306804	680.630057	680.630057	7049.248047	7049.248047	0.750000	0.750000
alfa CGbAIS	-15.000000	-15.000000	24.306209	24.307050	680.630057	680.630058	7049.251377	7049.251812	0.750000	0.750000

A estratégia do cluster adotada nestas simulações é diferente da proposta anteriormente, onde os indivíduos são agrupados pelas distâncias entre suas variáveis. Aqui, foi considerado somente o valor da função objetivo. Embora esta estratégia seja mais elitista que a original, há uma economia de tempo durante o processo de clusterização.

Um estudo preliminar indicou que o algoritmo não é muito sensível a alguns parâmetros. Assim, utilizando este método, diferente do proposto em [40], cada indivíduo tem seu próprio conjunto de parâmetros gerados no início. É possível concluir que o algoritmo proposto tem a mesma capacidade de encontrar o ótimo global que os demais. Entretanto, dois pontos devem ser considerados.

O primeiro, é que utilizando o controle de maturação a população rapidamente reduziu de 200 para 6 indivíduos, fornecendo maior velocidade ao algoritmo. Para os problemas G1, G7, G9 e G10, enquanto o α-simplex convergiu com aproximadamente 380.000 iterações, o CGbAIS convergiu com 290.000. Já para o problema G11 o α-simplex teve 310.000 iterações, enquanto o CGbAIS alcançou o resultado ótimo com somente 57.000 iterações.

O segundo ponto é que os problemas testados têm apenas uma situação ótima global, privilegiando métodos elitistas [40]. Mesmo assim, o CBbAIS apresentou ótimos resultados, provando a flexibilidade do algoritmo de otimização.

Porém, em diversas áreas, como economia, estudos sobre transmissão elétrica, apenas uma solução para o problema de otimização não é o bastante. Nestes casos o αCGbAIS é mais indicado que os métodos elitistas.

6.7.1 Testes e Resultados do α-CGbAIS para Sistemas Elétricos

Dois dos problemas mais comuns em algoritmos evolucionários são a sensibilidade para ajustar o parâmetro e o esforço computacional para assegurar a convergência. Para avaliar a eficiência do algoritmo proposto em tais situações dois cenários serão considerados: tempo computacional e sensibilidade de variação.

Para a análise será utilizado um problema de redução de perda com instalação de shunt em um conjunto pré-definido de barras.

$$minimize \sum_{(m,n)\in SBUS} P_{m,n} + P_{n,m} \tag{6.21}$$

Onde, SBUS é o conjunto de barras selecionadas para instalação de shunts.

O sistema é o IEEE-18 barras com limite de tensão que variam de 0.95 pu a 1.05 pu. Todos os limites da potência reativa são também considerados. O valor máximo para instalação dos shunts é 0.4 pu e a escolha das barras são as com as piores tensões, ie: 75, 117, 52, 51, 73, 57, 54, 50, 104, 105. Os parâmetros de simulação são: *npop* = 100, *nbest* = 50, *nworst*=20, *MaxInt*=40, *clustarAt*=5, *pm*= 0,05, *adl*=0,1, α=1,5 e *b*=0,2.

Para comparar os resultados dois outros métodos baseados em população são utilizados: o *Swarm* (evolutionry particle swarm optimization - EPSO), apresentado em [53] e o *differential evolution* (DE), apresentado em [45]. Para ambos os algoritmos o α-constrained mostrou melhores resultados. A tabela 6.5 apresenta os resultados das simulações após 30 iterações com 40 gerações. Os melhores resultados estão em negrito.

Como o algoritmo proposto consiste de três estratégias (α-constrained, clusterização e vetor gradiente), é interessante identificar a relevância de cada estratégia separado e do algoritmo final. Quatro simulações foram realizadas. A primeira mostra todas as características. A segunda, o α-constrained não é considerado, logo, alfa foi definido com valor 1. Na terceira, o receptor de edições (RE) não foi utilizado, e na quarta o processo de clusterização e o gradiente (CG) foram desabilitados.

Tabela 6-5 Resultados com α-CGbAIS (e variações), EPSO e DE: Sistema IEEE-118 barras

Status	α*CGbAIS*		*Alfa = 1*		*RE OFF*		*CG OFF*		*aCGbAIS**		*EPSO*		*Dif. Evolution*	
Pop. Size	*10*	*30*	*10*	*30*	*10*	*30*	*10*	*30*	*10*	*30*	*10*	*30*	*10*	*30*
Best	1.9272	**1.9271**	1.9274	1.9274	1.9272	1.9272	1.9289	1.9278	1.9272	1.9272	1.9272	**1.9271**	1.9272	**1.9271**
Worst	1.9273	**1.9272**	1.9295	1.9290	1.9277	1.9378	1.9326	1.9312	1.9305	1.9304	1.9275	1.9273	1.9490	1.9288
Mean	1.9272	**1.9272**	1.9281	1.9278	1.9274	1.9273	1.9300	1.9294	1.9281	1.9274	1.9273	**1.9272**	1.9375	1.9275
Time (s)	6.75	**7.59**	6.65	7.53	14.76	15.02	1.84	2.41	4.15	4.48	48.34	142.91	21.62	73.15

Após analisar estas quatro simulações, as conclusões são:

- o α-constrained alcança uma boa precisão ao final do processo. Observe que os valores b*est*, *worst* e *mean* são melhores para o α-CGBAIS.
- Utilizando somente a estratégia CG, os resultados são bons, mas o tempo computacional é alto. Como o algoritmo evolui utilizando somente utilizando o gradiente, é esperado pequenas melhoras para o ótimo local. Isto explica os resultados mais altos (*mean* e *worst*) comparados com o algoritmo completo α-CGBAIS.
- Já utilizando somente o receptor de edições as respostas são mais rápidas. Mas a precisão é menor, por isto não são seguros.

Com todas estas conclusões prévias é possível deduzir que o receptor de edições é responsável pelo tempo computacional, mas não pela precisão e convergência, enquanto que o gradiente e o α*–constrained* são responsáveis pelas melhores soluções. Com base nestes conhecimentos uma quinta simulação foi realizada (α-CGbAIS*). Neste caso, o receptor de

edições e o α–*constrained* são aplicados até o melhor que o melhor indivíduo presente tenha um nível de satisfação igual a 1. Até este ponto, o processo de clusterização reduz a população e o gradiente é usado de acordo com a proposta inicial. Este procedimento teve um melhor tempo computacional, com as melhores soluções iguais ao α-CGbAIS. Porém os valores *mean* e *worst* provam esta abordagem não é tão segura quanto a original (α-CGbAIS).

Comparando o α-CGbAIS com os métodos *EPSO* e com DE conclui-se que o método proposto é mais rápido e confiável.

7 Resultados Relativos a Modelagem via Orientação a Aspectos

Para medir os benefícios da modelagem de um FPO utilizando Orientação a Aspectos, é necessário observar dois pontos. O primeiro refere-se às métricas de software, isto é, utilizar índices de qualidade de sistemas computacionais para avaliar os benefícios do AOP em relação a outras formas de modelagem. O segundo ponto é a flexibilidade prática, ou seja, como um sistema baseado em AOP pode atender de forma melhor a um operador do sistema. Estes dois pontos são exibidos a seguir.

7.1 Métricas de Software

As métricas de software avaliam a qualidade de um sistema baseado em um conjunto de regras bem definido. A literatura apresenta diversos índices focando a medida de qualidade, como por exemplo, complexidade, custo de manutenção, escalonamento, modularidade, esforço, e outros.

Esta seção apresenta a análise de desempenho da solução final apresentada por cada paradigma de desenvolvimento utilizando algumas métricas. Os índices utilizados são: *linhas de código fonte, complexidade cliclomática e Halstead* [54].

O índice referente à *linha de código fonte*, (*SourceLines of Code - SLOC*), é uma métrica utilizada para avaliar o número de código em um programa computacional. Ela

considera o tamanho do software que foi produzido. É usualmente empregada para estimar a quantidade de esforço necessário para desenvolver um programa e também para avaliar a produtividade e manutenção uma vez que o sistema está pronto.

Já a *complexidade cliclomática* fornece uma medida quantitativa da complexidade lógica de um programa. Esta métrica mede diretamente o número de caminhos independentes (qualquer caminho ao longo do programa que introduza pelo menos um novo conjunto de instruções de processamento ou uma nova condição) através de um código de um programa. Em temos de grafo de fluxo, um caminho independente deve incluir pelo menos uma aresta que não tenha sido atravessada antes de o caminho ser definido, ou seja, um caminho independente não pode ser formando por uma combinação de caminhos já especificados. Dentre as formas de calcular a complexidade cliclomática *V(G),* para um grafo de fluxo, *G*, temos a equação :

$$V(G) = E - N + 2 \tag{7.1}$$

Onde *E* é o número de arestas.

Finalmente, a *complexidade de Halstead*, mede a complexidade do código através da análise dos operadores de cada módulo. Esta métrica fornece medidas singulares de um software de computador. É um forte indicador da manutenabilidade e é muito empregada para estimar custos de manutenção de software.

A métrica utiliza um conjunto de medidas primitivas que podem ser originadas após o código ser gerado, ou estimadas quando o projeto é completado. As medidas são:

n1= numero de operadores distintos que aparecem em um programa (*if, =, while, etc*..)

n2= número de operandos distintos que aparecem em um programa *(variávies ou constantes)*

N1= número total de ocorrência operadores (n1)

N2= número total de ocorrência operandos (n2)

Esta métrica tem cinco medidas derivadas: comprimento do programa ou program length (PL – Equação7. 2) , vocabulário do programa ou "*program vocabulary*" (PV- Equação 7.3), *volume* (V- Equação 7.4), *dificuldade* ou "*difficult*" (D – Equação 7.5) e *esforço* ou "*effort*" (E - Equação7.6).

$$PL = N_1 + N_2 \tag{7.2}$$

$$PV = n_1 + n_2 \tag{7.3}$$

$$V = N * (\ln (PV)) \tag{7.4}$$

$$D = \frac{n_1}{2} * \frac{n_2}{2} \tag{7.5}$$

$$E = D * V \tag{7.6}$$

Para demonstrar o impacto do paradigma AOP sobre os outros métodos alguns resultados são apresentados. O número de variáveis (VAR), métodos de otimização (OM) e função objetivo (OB) são mostrados na Tabela 7-1 para os paradigmas Estruturado, Orientado a Objetos e Orientado a Aspectos. Note que, no que diz respeito a complexidade, a programação Estruturada apresenta um crescimento exponencial, a POO linear e na Orientação a Aspectos este índice fixo. Outra importante observação é fornecida pelo índice *Ciclomático*. Enquanto a programação Estruturada e a POO são dependentes do número de variáveis e funções, a Orientação a Aspecto é fixa em um mínimo, dispensando todo o tratamento lógico presente nos outros paradigmas.

Tabela 7-1 Resultados das Métricas de Software

	SM	OOM	AOM
SLOC	om.ob.var.20	10.ob + 10.var + 5.om	60
Cyclomatic	om.ob.var + 2	om + ob + var -1	3

Considerando um exemplo prático, onde existem 3 métodos de otimização, 4 funções objetivo e 5 variáveis, o esforço final de desenvolvimento de cada paradigma é mostrado na Tabela 7-2.

Tabela 7-2 - Métricas de Software com om=3, ob=4, var=5

	SM	OOM	AOM
SLOC	1,200	105	60
Cyclomatic	62	11	3
PL	1,560	143	109
PV	38	45	74
V	5,675	544	469
D	2,400	462	457
E	13,619,123	251,491	214,516

Temos que considerar que estes dados são somente para a configuração mencionada acima (onde desejo utilizar 3 métodos de otimização, 4 funções objetivo e 5 variáveis). Porém, se deseja alterar qualquer outro dado no sistema, para o paradigma aop estes valores praticamente se mantêm, já para o orientado a objetos e estruturado estes valores serão bem diferentes (principalmente para o estruturado).

Na Figura 7.1, mostra um gráfico com os sete casos estudados (o gráfico esta em escala logarítmica). O número de variáveis (VAR), método de otimização (OM) e função objetivo (OB) são apresentados a seguir para cada caso. É possível concluir que, a medida que a complexidade cresce, a modelagem orientada a aspectos (AOM) é mais apropriado, uma vez que sua complexidade de desenvolvimento é fixa. Em situações onde apenas um método de otimização com poucas funções objetivo e controles estão disponíveis, a metodologia OOM é mais apropriada. Finalmente a programação estruturada (PS) é indicada apenas em situações muito simples.

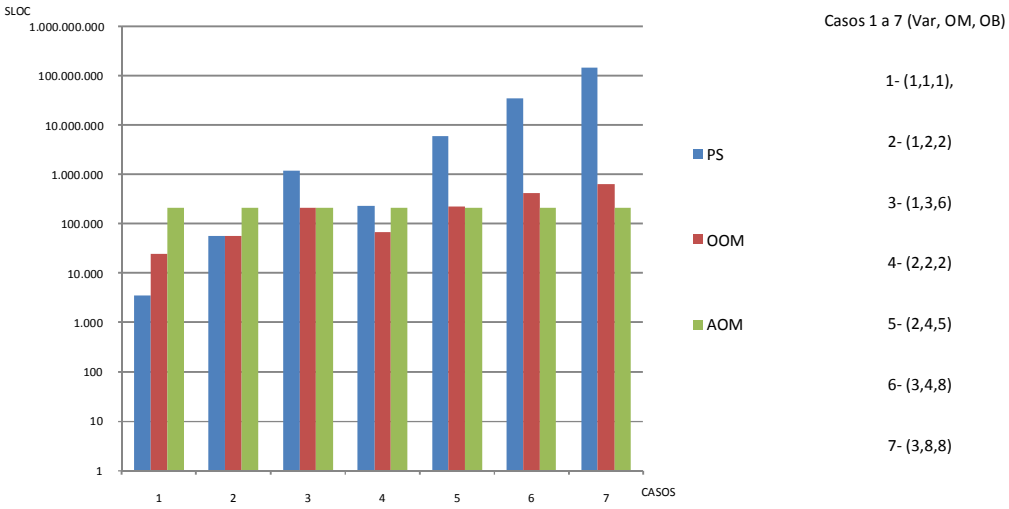

Figura 7.1- Comparação entre as Métricas de Software

Em uma comparação final entre OO e AOP é possível inferir que um modelo mais abstrato é capaz de representar melhor o problema proposto mesmo em casos que um maior número de linhas de código for gerado. Isto se deve ao fato de que, se a complexidade do sistema crescer, com a adição de novas funcionalidades, a orientação a aspecto consegue manter a complexidade do software sob controle.

7.2 Modularidade

Esta seção avalia duas propriedades distintas em um software: o resultado das simulações numéricas e a flexibilidade. Como já mencionado, o paradigma AOP não interfere nos resultados numéricos ou no tempo computacional. Conseqüentemente, os índices aqui representam os métodos de otimização (BFHS E CONJ) e a qualidade na engenharia do software.

Dada a impossibilidade de mostrar todas as combinações fornecidas pelo sistema, as simulações irão focar dois dos maiores problemas em sistemas elétricos de potência: sobrecarga na linha de transmissão e minimização da perda. Os tipos individuais de ações são VMC, APG e

ACL. Também foi considerada uma ação composta com VMC, TAP e APG. Estes controles ficam disponíveis para o sistema, que escolherá quais devem ser empregados. No total são geradas 23 combinações possíveis (4!-1). Os métodos BFHS e CONJ serão usados, totalizando aproximadamente 100 combinações (2x(3+23)x2). Para cada combinação, a ordem do controle de ativação e o resultado final são exibidos.

Para avaliar o impacto de AOP sobre o problema do FPO iremos considerar inicialmente a flexibilidade. Este índice refere-se a interconexão entre os módulos do sistema, isto é, o impacto que a alteração em uma determinada classe tem sobre o restante do sistema.

A Tabela 7.3 mostra as classes onde as linhas tiveram de ser geradas ou alteradas para gerar cada combinação do FPO. Os resultados indicam que as classes do aspecto não tiveram alterações quando novas funções objetivo ou ações de controle foram desenvolvidas no aspecto do fluxo de potência.

Tabela 7-3 Impacto no Desenvolvimento de Novas Ações de Controle e Funções Objetivo

Classes	Controls and Functions													
	ACL	VMS	OVL	LSS	APF	RPF	GRC	RCL	VMC	SHT	APG	RPG	FCT	TAP
Cls_PowerFlow	15	.	20	20	.	.	.	.	3	.	.	.	.	.
Cls_Buses	7	3	.	.	.	.	3	7	11	5	5	7	.	.
Cls_Lines	.	.	3	3	3	3	.	.	.	.	.	.	7	10
TOTAL	*22*	*3*	*23*	*23*	*3*	*3*	*3*	*7*	*14*	*5*	*5*	*7*	*7*	*10*

Para testar a capacidade de convergência, os sistemas CTEEP e IEEE18 foram utilizados. A Tabela 7.4 mostra a sobrecarga nas linhas de transmissão e suas perdas.

Tabela 7-4 Principais Sobrecargas e Perdas nas Linhas de Transmissão

System	ID	NF	NT	LOSS	FLIMIT	OVL
IEEE118	7	8	5	0.00	3.00	0.47
IEEE118	94	49	66	0.44	99.00	0.00
CTEEP	361	674	1917	0.00	0.80	0.44
CTEEP	119	538	559	0.38	15.24	0.00

Para o processo de simulação, duas funções objetivo de minimização foram escolhidas: sobrecarga da linha 361 e perda elétrica da linha 119 (CTEEP); e sobrecarga da linha 7 e perda elétrica da linha 94 (IEEE 118). De acordo com o algoritmo apresentado na seção 6.5.3. O próximo passo é aplicar a função heurística e identificar as melhores ações a serem tomadas. As Tabelas 7.5 e 7.6 mostram os controles IDs (BID para controles conectados as barras e LID para os controles conectados as linhas) e suas respectivas sensibilidades, aplicadas para este fim.

Tabela 7-5 Sensibilidades de Sobrecarga das Linhas de Transmissão

CTEEP - Line 361								IEEE 118 - Line 7							
APG		ACL		TAP		VMC		APG		ACL		TAP		VMC	
BID	*Sen*	*BID*	*Sen*	*LID*	*Sen*	*BID*	*Sen*	*BID*	*Sen*	*BID*	*Sen*	*LID*	*Sen*	*BID*	*Sen*
519	-14.3	**1917**	66.9	**121**	32.1	**501**	-65.7	**12**	-0.32	**8**	0.16	**35**	0.25	**8**	22.02
518	-13.2	**665**	19.7	**142**	4.4	**502**	-26.8	**26**	0.01	**10**	0.15	**50**	0.09	**4**	2.15
517	-7.5	**675**	17.7	**217**	3.5	**533**	-22.9	**10**	0.15	**26**	0.01	**97**	0.02	**12**	1.02
511	-2.0	**638**	16.2	**158**	-4.2	**400**	-14.7	**8**	0.15	**12**	-0.32	**7**	-0.93	**6**	0.28

Tabela 7-6 Sensibilidades de Perdas das Linhas de Transmissão

CTEEP - Line 119								IEEE 118 - Line 94							
APG		ACL		TAP		VMC		APG		ACL		TAP		VMC	
BID	*Sen*	*BID*	*Sen*	*LID*	*Sen*	*BID*	*Sen*	*BID*	*Sen*	*BID*	*Sen*	*LID*	*Sen*	*BID*	*Sen*
525	-2.4	**462**	3.32	**217**	4.72	**501**	-74.2	**49**	-0.08	**49**	0.09	**97**	0.40	**54**	0.36
526	-2.1	**461**	3.03	**34**	4.03	**502**	-30.2	**46**	-0.08	**48**	0.08	**50**	0.06	**46**	0.26
400	-2.0	**459**	3.02	**35**	4.03	**533**	-24.2	**42**	-0.06	**46**	0.08	**89**	0.01	**42**	0.22
406	-2.0	**455**	2.72	**104**	4.01	**400**	-16.7	**66**	0.06	**45**	0.08	**91**	-0.01	**65**	-0.44

Os dados mostrados por estas tabelas indicam quais controles são mais eficazes para cada categoria. Deve ser mencionado que diferentes ações de controle têm diferentes impactos no sistema. Porém, uma alta sensibilidade nem sempre indica que um dado controle é mais indicado para solucionar um problema específico que outro. Por exemplo, as Tabelas 7.5 e 7.6 mostram que VMC tem um valor médio de sensibilidade alto, pré-indicando que este tipo de ação deveria ter um impacto mais significativo. Porém, como a tensão em uma dada barra pode variar de 0.95 pu e 1.05 pu, o controle tem uma escala limitada, significando um impacto

limitado no sistema. Por outro lado, utilizar outro tipo de controle com escala maior, como corte de carga, oferece um impacto mais significativo na função analisada.

Para processar o problema de otimização, quatro ações de controle para cada sistema foram escolhidas (*ids* em negrito nas Tabelas 7.5 e 7.6). O critério, além da sensibilidade, foi de adotar controles com escala disponível. Esta escala é indicada pelo controle do ponto de operação atual, seus limites e a direção da sensibilidade: positiva (elevando o controle eleva o objetivo) e negativo (elevando o controle diminui o objetivo).

As simulações, para os tipos de controles apresentados pelas Tabelas 7.7 e 7.8, comparam um conjunto com oito controles aplicados para redução de perda e sobrecarga. O resultado mostra que, exceto a minimização de sobrecarga usando VMC e TAP como ações de controle, as saídas numéricas são quase sempre as mesmas, também mostra que CONJ apresenta melhor tempo computacional.

Tabela 7-7 Resultados das Ações Isoladas no Sistema CTEEP

Controls					Function	Initial	BFHS			CONJ		
Type					MIN	Value	Final V.	%	Time	Final V.	%	Time
VMC	400	501	502	533	OVL	0.44	0.41	5.05	9.59	0.41	5.05	4.45
ACL	638	665	675	1917	OVL	0.44	0.00	100.00	10.44	0.00	100.00	5.23
APG	511	517	518	519	OVL	0.44	0.18	58.49	24.13	0.18	58.49	8.67
TAP	121	142	158	217	OVL	0.44	0.39	10.78	18.88	0.39	10.78	7.14
VMC	400	501	502	533	LOSS	0.38	0.35	6.65	10.44	0.35	6.65	4.72
ACL	455	459	461	462	LOSS	0.38	0.34	10.90	27.70	0.33	11.70	16.52
APG	400	406	525	526	LOSS	0.38	0.35	6.38	15.06	0.35	6.38	5.70
TAP	104	105	106	217	LOSS	0.38	0.37	1.06	10.36	0.37	1.06	7.45

Tabela 7-8 Resultados das Ações Isoladas no Sistema IEEE

Controls					Function	Initial	BFHS			CONJ		
Type					MIN	Value	Final V.	%	Time	Final V.	%	Time
VMC	8	4	12	6	OVL	0.47	0.41	12.55	2.85	0.33	29.61	0.35
ACL	8	10	26	12	OVL	0.47	0.00	100.00	3.25	0.00	100.00	0.64
APG	12	26	10	8	OVL	0.47	0.00	100.00	5.38	0.00	100.00	2.45
TAP	35	50	97	7	OVL	0.47	0.44	7.21	1.68	0.47	0.00	1.65
VMC	54	46	42	65	LOSS	0.44	0.42	5.00	1.33	0.42	4.55	0.46
ACL	49	48	46	45	LOSS	0.44	0.23	47.73	5.94	0.23	47.73	2.73
APG	49	46	42	66	LOSS	0.44	0.15	67.05	5.39	0.15	67.05	4.78
TAP	97	50	89	91	LOSS	0.44	0.42	3.64	1.45	0.42	4.09	0.82

Para a minimização de sobrecarga todas as ações relativas ao objetivo tiverem o mesmo valor final.

É necessário apresentar algumas considerações:

1. ACL e APG alcançaram 100% de redução de sobrecarga (IEEE 118) para ambos os métodos de otimização.
2. TAP não tem nenhum efeito na minimização de sobrecarga usando CONJ (IEEE 118)
3. O método VMC foi o controle mais eficiente para minimização da perda e da sobrecarga, indicando esta ação como uma possível alternativa para ajustar pequenas irregularidades no sistema.

Apesar do ACL e APG terem apresentados os melhores resultados para evitar a sobrecarga, geralmente estes controles são os últimos a serem indicados. O sistema freqüentemente opera utilizando controles shunt (não simulados aqui), TAP, VMC e então APG e ACL. A função para minimizar a perda tem duas ações que não são consideradas em situações reais: ACL e APG. Estas ações foram simuladas com o objetivo de demonstrar a flexibilidade do sistema.

Finalmente, uma ação composta é exibida pela Tabela 7.9. Para esta simulação as ações APG, VMC e TAP, mostradas na Tabela 7.6, foram usadas para o sistema IEEE 118 focando a redução de perda. Os métodos CONJ e BFHS foram testados. Os dados da Tabela 7.9 mostram que para o cenário de minimização da perda ambas as estratégias de otimização alcançaram mais de 70% de redução de perda, com tempo de convergência de 18, 047 para BHFS e 13, 063 para CONJ.

Tabela 7-9 Resultado das Ações Múltiplas

Method	Initial Value	Final Value	%	Time
BFHS	0.440	0.128	70.909	18.047
CONJ	0.440	0.125	71.591	13.063

8 Conclusão

Este trabalho abordou um tema importante que é desenvolvimento de sistemas computacionais para otimização de fluxo de potência. Duas estratégias distintas foram analisadas. Primeiro uma nova técnica de desenvolvimento de sistemas, AOP, foi utilizada para um desenvolvimento mais robusto do sistema computacional, gerando um código mais fácil de ser mantido e evoluído. O outro aspecto considerado foi o desenvolvimento de uma variante de uma metaheurística já existente na literatura, o AIS. Com os conceitos propostos por este trabalho (uso do gradiente, clusterização de indivíduos e alfa- Constrained) foi possível gerar um método de otimização mais robusto e rápido. Utilizando os dois conceitos em uma ferramenta, foi demonstrado índices de versatilidade e confiabilidade com resultados aprimorados.

9 Bibliografia

1 L. Honório, D. Barbosa, A. Zambroni de Souza, and C. Lopes. "Intelligent Optimal Power Flow System Development Using Aspect-Oriented Modeling". *Power Systems, IEEE Transactions on*, 22, 4 (2007), 1826–1834.

2 B. Hakavik, A. Holen. "Power system modelling and sparse matrix operations using object-oriented programming". *Power Systems, IEEE Transactions on*, 9, 2 (1994), 1045–1051.

3 A. Manzoni. "Power systems dynamics simulation using object-oriented programming,"". *IEEE Transactions on Power Systems*, 14, 1 (1999), 249–255.

4 I. Drezga, R. Broadwater, A. Sugg, E. Design, and B. Inc. "Object-oriented analysis of distribution system recon…guration forpower restoration". *Power Engineering Society Summer Meeting, 2001. IEEE,* 2 (2001).

5 A. Rodriquez. "Applying use cases, object-oriented analysis, and UML in the development of power scheduling systems". *Power Engineering Society Winter Meeting,2002. IEEE*, 1 (2002).

6 Yamamoto, J. Suzuki and Y. "Extending UML with Aspects: Aspect Support in the Design Phase". *LECTURE NOTES IN COMPUTER SCIENCE* (1999), 299-299.

7 Brichau, K. Gybels and J. "Arranging language features for more robust pattern-based crosscuts". *in Proceedings of the 2nd international conference on Aspect-oriented software development* (2003), 60–69.

8 J. Irwin, J. Loingtier, C. Lopes, C. Maeda, A. Mendhekar, J. Lamping, and G. Kiczales. "Aspect-oriented programming". *Lect. Notes Comp. Sci*, 1241, 220-242.

9 G. Torres, V. Quintana. "On a nonlinear multiple-centrality-corrections interior-point method for optimal power ‡ow". *Power Systems, IEEE Transactions on*, 16, 2 (2001), 222-228.

10 M. Liu, S. Tso, and Y. Cheng. "An extended nonlinear primal-dual interior-point algorithm

for reactive-power optimization of large-scale power systems with discrete control variables". *Power Systems, IEEE Transactions on*, 17, 4 (2002), 982-991.

11 L. Honorio, A. Leite da Silva, and D. Barbosa. "A Gradient-Based Immune System Applied to Optimal Power Flow Problems". *LECTURE NOTES IN COMPUTER SCIENCE*, 4628 (2007), 1-12.

12 S. Ishak, A. Abidin, and T. Rahman. "Static Var compensator planning using artificial immune system for loss minimisation and voltage improvement". *in Power and Energy Conferencee, 2004. PECon 2004. Proceedings. National,* (2004), 41-45.

13 P. Biskas, N. Ziogos, A. Tellidou, C. Zoumas, A. Bakirtzis, and V. Petridis. "Comparison of two metaheuristics with mathematical programming methods for the solution of OPF". *in Generation, Transmission and Distribution, IEE Proceedings-*, 153, 1 (2006), 16-24.

14 A. Esmin, G. Lambert-Torres, and A. de Souza. "A hybrid particle swarm optimization applied to loss power minimization IEEE Transactions on Power Systems". *IEEE Transactions on Power Systems*, 20, 2 (2005), 859-866.

15 N. Sinha, R. Chakrabarti, and P. Chattopadhyay. "Fast evolutionary programming techniques for short-term hydrothermal scheduling". *Electric Power Systems Research*, 66 (2003), 97-103.

16 S. Kannan, S. Slochanal, and N. Padhy. "Application and comparison of metaheuristic techniques to generation expansion planning problem". *IEEE Transactions on Power Systems*, 20, 1 (2005), 466-475.

17 H. Chen, X.Wang. "Cooperative coevolutionary algorithm for unit commitment". *Power Systems, IEEE Transactions on*, 17, 1 (2002), 128-133.

18 M. Tahk, B.Sun. "Coevolutionary augmented Lagrangian methods for constrained optimization". *Evolutionary Computation, IEEE Transactions on*, 4, 2 (2000), 114-124.

19 Y. Zhou, J.He. "A Runtime Analysis of Evolutionary Algorithms for Constrained Optimization Problems". *IEEE Transactions on Evolutionary Computation*, 11, 5 (2007), 608-619.

20 Abido, M. "A niched Pareto genetic algorithm for multiobjective environmental/economic dispatch". *International Journal of Electrical Power and Energy Systems*, 25, 2 (2003), 97-105.

21 L. de Castro, F. Von Zuben. "Learning and optimization using the clonal selection principle".

Evolutionary Computation, IEEE Transactions on, 6, 3 (2002), 239-251.

22 G. Liao. "Application of an immune algorithm to the short-term unit commitment problem in power system operation". *in Generation, Transmission and Distribution IEE Proceedings-*, 153, 3 (2006), 309-320.

23 T. Rahman, S. Suliman, and I. Musirin. "Artificial Immune-Based Optimization Technique for Solving Economic Dispatch in Power System". *LECTURE NOTES IN COMPUTER SCIENCE*, 3931 (2006), 338.

24 P. Kundur. "Power System Stability and Control". *McGraw-Hill Professional* (1994).

25 A. de Souza, L. Honorio, G. Torres, and G. Lambert Torres. "Increasing the Loadability of Power Systems Through Optimal Local Control Actions". *IEEE Transactions on Power Systems*, 19, 1 (2004), 188-194.

26 D.Lawden. "Analytical Methods of Optimization". *Hafner Press* (1975).

27 F. Fabozzi, P. Kolm, D. Pachamanova, and S. Focardi. "Robust Portfolio Optimization". *JOURNAL OF PORTFOLIO MANAGEMENT*, 33, 3 (2007), 40.

28 G. Cornuejols, R. Tütüncü, and MyiLibrary. *"Optimization methods in finance"*. Cambridge University Press, 2007.

29 D. Luenberger. *Linear and Nonlinear Programming*. Springer, 2003.

30 Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda,Cristina Lopes, Jean-Marc Loingtier and John Irwin. A s p e c t - O r i e n t e d P r o g r a m m i n g. *ECOOP '97 (LNCS 1241)* (1997), 220.

31 Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold. An Overview of AspectJ. *Lecture Notes in Computer Science*, 2072 (2001), 327-354.

32 Cristina Lopes, Gregor Kiczales. "Recent Developments in AspectJ". *In Proc. of AOP workshop at ECOOP'98, Workshop Reader, Springer-Verlag LNCS n. 1543* (1998).

33 Ivar Jacobson, Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases.* Addison Wesley Professional, 2004.

34 Siobhán Clarke, Elisa Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach.* Addison Wesley Professional, 2005.

35 LADDAD, RAMNIVAS. *AspectJ in Action PRACTICAL ASPECT-ORIENTED*

Programming. MANNING, 2003.

36 L. De Castro, F. Von Zuben. *"Artificial Immune Systems: Part I -Basic Theory and Applications"*. 1999.

37 J. Hunt, D.Cooke. "Learning using an artificial immune system". *Journal of Network and Computer Applications*, 19, 2 (1996), 189-212.

38 W. Sheng, S. Swift, L. Zhang, and X. Liu. "A weighted sum validity function for clustering with a hybrid niching genetic algorithm". *Systems, Man and Cybernetics Part B, IEEE Transactions on*, 35, 6 (2005), 1156-1167.

39 J. Timmis, C.Edmonds. "A Comment on Opt-AiNET: An Immune Network Algorithm for Optimisation". *Lecture Notes in Computer Science* (2004), 308-317.

40 T. Takahama, S.Sakai. "Constrained optimization by applying the/spl alpha/constrained method to the nonlinear simplex method with mutations". *Evolutionary Computation, IEEE Transactions on*, 9, 5 (2005), 437-451.

41 K. Almeida, R.Salgado. "Optimal power flow solutions under variable load conditions". *Power Systems, IEEE Transactions on*, 14, 4 (2000), 1204-1211.

42 J. Dennis Jr, R.Schnabel. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16)" (1996).

43 T. Runarsson, X.Yao. "Stochastic ranking for constrained evolutionary optimization". *Evolutionary Computation, IEEE Transactions on*, 4, 3 (2000), 284-294.

44 Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., and Nakanishi, Y. "A particle swarm optimizatoin for reactive power and voltage control considering voltage security assessment". *IEEE Trans. on Power Systems*, 15, 4 (2000), 1232-1239.

45 Wang, Z., Chung, C.Y., Wong, K.P., and Tse, C.T. "Robust power System stabilizer design under multi-operating conditions using differential evolution". *IET Generation, Transmission and Distribuition*, 2, 5 (2008), 690-700.

46 Chang, C-F., Wong, J-J., Chiou, J-P., Su, C-T. "Robust searching hybrid differential evolution method for optimal reactive power planning in large-scale distribution systems". *Electrical Power Systems Reaserch*, 77 (2007), 430-437.

47 L. de Castro, J.Timmis. "An artifiial immune network for multimodal function optimization". *in Proceedings of IEEE Congress on Evolutionary Computation (CEC'02)*, 1 (2002), 699-674.

48 Croes, G. "A method for solving traveling salesman problems". *Operations Research*, 6, 6 (1958), 791-812.

49 S. Lin. "Computer solution of the traveling salesman problem". *Syst. Tech. J*, 44 (1965), 2245-2269.

50 M. Dorigo, L.Gambardella. "Ant colony system: a cooperative learning approach to the travelingsalesman problem". *Evolutionary Computation, IEEE Transactions on*, 1, 1 (1997), 53-66.

51 Y. Zhu, S. Gao, H. Dai, F. Li, and Z. Tang. "Improved Clonal Algorithm and Its pplication to Traveling Salesman Problem". *IJCSNS,* 7, 8 (2007), 109.

52 Holland, J. "Genetic Algorithms and the Optimal Allocation of Trials". *SIAM Journal on Computing*, 2 (1973), 88.

53 Keko, H., Duque, A.J., and Miranda, V. "A Multiple Scenario Security Constrained Reactive Power Planning Tool Using EPSO". *Int. Conf. on Intelligent System Application to Power Systems, ISAP, 2007* (2007), 1-6.

54 Pressman, Roger S. *Engenharia de Software*. 2006.

55 G. Kiczales. "Aspect-Oriented Programming". *Computing Surveys 28, 4es* (1996), 154.