TESE DEFENDIDA POR FLÁVIA Pizzi

E APROV

COMISSÃO JULGADORA EM 27 02 20

R Pavanello

ORIENTADOR

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Otimização Topológica de Estruturas Utilizando Algoritmos Genéticos.

Autor: **Flávia Pizzirani**
Orientador: **Renato Pavanello**

12/03

Universidade Estadual de Campinas
Faculdade de Engenharia Mecânica
Comissão de pós-graduação em engenharia mecânica
Departamento de Mecânica Computacional

Otimização Topológica de Estruturas Utilizando Algoritmos Genéticos.

Autora: **Flávia Pizzirani**
Orientador: **Renato Pavanello**

Curso: Engenharia Mecânica.
Área de Concentração: Mecânica dos Sólidos e Projeto Mecânico.

Dissertação de mestrado apresentada à comissão de Pós-Graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Mestre em Engenharia Mecânica.

Campinas - 2003
S.P. - Brasil

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

BID 295216

P689o

Pizzirani, Flávia
Otimização topológica de estruturas utilizando algoritmos genéticos / Flávia Pizzirani.--Campinas, SP: [s.n.], 2003.

Orientador: Renato Pavanello.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Algoritmos genéticos. 2. Método dos elementos finitos. 3. Otimização estrutural. I. Pavanello, Renato. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

DISSERTAÇÃO DE MESTRADO

Otimização Topológia de Estruturas Utilizando Algoritmos Genéticos

Autora: **Flávia Pizzirani**
Orientador: **Renato Pavanello**

Prof. Dr. Renato Pavanello, Presidente
DMC/FEM/UNICAMP

Prof. Dr. Eurípedes Guilherme de O. Nobrega
DMC/FEM/UNICAMP

Profa. Dra. Vera Lucia da Rocha Lopes
DMC/FEM/UNICAMP

Campinas, 27 de fevereiro de 2003.

Dedicatória

Dedico este trabalho aos meus pais, Jayme e Cidinha, que sempre me apoiaram e me incentivaram em todos os momentos de minha vida, principalmente nos estudos, onde fizeram de tudo para que eu chegasse até aqui.

Dedico também ao meu marido Luciano, que me ajudou e me apoiou nas horas difíceis e aos meus irmãos, Fabiane e Guilherme, cuja amizade me são muito valiosas.

Agradecimentos

Gostaria de agradecer ao meu orientador Prof. Renato Pavanello, pela dedicação e pela amizade, que tornou a execução deste trabalho muito prazerosa.

Agradeço à Prof[a]. Véra Lucia da Rocha Lopes, que me apresentou a um trabalho pelo qual me apaixonei: a pesquisa; agradeço também pela amizade e pelo incentivo.

Agradeço também ao Prof. Janito Vaquero Ferreira pelas dicas valiosas em C++ e pela ajuda com o *MefLab++*.

Agradeço aos colegas do DMC pelas risadas e pelo companheirismo.

Agradeço ao CNPq pelo financiamento deste projeto.

Resumo

Pizzirani, Flávia, *Otimização de Estruturas Utilizando Algoritmos Genéticos*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2003, 104 pp. Dissertação(Mestrado)

Este trabalho apresenta uma investigação sobre a aplicação de Algoritmos Genéticos em otimização estrutural. A otimização topológica de estruturas reticuladas e de problemas de estado plano de tensões é realizada usando-se Algoritmos Genéticos. Os Algoritmos Genéticos são mecanismos de busca baseados na teoria da Seleção Natural de Charles Darwin e nos mecanismos da genética. Estes algoritmos trabalham com uma população de possíveis soluções que evoluem de acordo com operadores genéticos probabilísticos. Neste trabalho faz-se uma revisão bibliográfica sobre o uso dos Algoritmos Genéticos aplicados ao problema de otimização topológica. Realiza-se uma modelagem clássica em elementos finitos para análise linear estática das estruturas que serve de base para avaliar as funções de *fitness* do problema. O algoritmo implementado é baseado em uma codificação binária, que permite descrever a presença ou ausência de material em um determinado espaço de trabalho. O problema de minimização de massa, sujeito a restrições mecânicas de deslocamento e tensões máximas é resolvido usando-se um esquema de penalidades. São estudados vários parâmetros do algoritmo, tais como, repetibilidade, esquemas de penalização e tipos de *crossover*. No decorrer do trabalho são apresentados exemplos e ao final estão as conclusões e sugestões para trabalhos futuros na área.

Palavras Chave: Otimização Estrutural, Algoritmos Genéticos, Método dos Elementos Finitos.

Abstract

Pizzirani, Flávia, *Structural Optimization Using Genetic Algorithms*, Campinas: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2003, 104 pp. Dissertação(Mestrado)

This work presents an investigation about the aplication of Genetic Algorithms in structural optimization. The topological optimization of trusses and bidimensional elasticity problems is carried out using Genetic Algorithms. The Genetic Algorithms are search mechanisms based on Charles Darwin's Natural Selection Theory and in genetic mechanisms. These algorithms work with a population of possible solutions that evolve based on probabilistic genetic operators. In this work a bibliographical review about the use of Genetic Algorithms applied to the problem of topological optimization. A linear static analysis is done with the Finite Element Method for modelling structures. This analisys serves as a basis to evaluate the fitness functions of the problem. The implemented algorithm is based on a binary codification that allows describe the presence or absence of material in a determined workspace. The minimization of mass problem, when subjected to mechanical restrictions of displacement and maximum tensions, is decided using a scheme of penalties. Some parameters of the algorithm, such as repeatability, projects of penalization and types of crossover, are studied. In elapsing of the work some examples are presented and in the end some conclusions and suggestions for future works in the area are given.
Key Words: Structural Optimization, Genetic Algorithms, Finite Element Method

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Lista de Símbolos

Capítulo 1

Introdução

Durante muito tempo o projeto e a construção das estruturas foram feitos utilizando-se da experiência prática dos especialistas. As decisões eram - e muitas vezes ainda são - tomadas a partir de suposições e intuições de pessoas que ao longo do tempo adquiriram experiências na área de projeto e fabricação destas estruturas. De fato, este procedimento vem sendo usado por muitos, e com certo sucesso. A principal dificuldade é a da economia. Como realizar um projeto consistente sem desperdiçar tempo e material? Como realizar um projeto com a certeza de que ele não precisará ser reforçado posteriormente? É exatamente aí que os procedimentos de otimização podem ajudar. Os métodos de otimização estrutural auxiliam na criação de estruturas eficientes que satisfaçam as necessidades de maneira eficaz. A Figura 1.1 mostra de forma esquemática as diferenças entre um projeto convencional e um projeto otimizado.

A otimização de estruturas tem sido uma área ativa de pesquisa no campo da busca e otimização. Pode-se encontrar problemas envolvendo otimização de estruturas nas mais variadas áreas, desde o planejamento de telhados até construções de estações espaciais e painéis solares. Várias técnicas baseadas em métodos clássicos de otimização surgiram nos últimos anos, buscando encontrar estruturas ótimas para problemas até então sem solução. Existem três tipos de abordagens para o problema de otimização de estruturas: a otimização topológica, a otimização paramétrica e a otimização de forma. Neste trabalho aborda-se a otimização topológica de estruturas; uma breve explicação dos três tipos de abordagem será dada na seção 2.3. Para a resolução destes problemas existem muitos métodos de otimização, desde os mais antigos métodos de solução analítica, passando pelos métodos iterativos e mais recentemente os métodos de otimização evolutivos e meta-heurísticos.

Os Algoritmos Genéticos, são métodos evolutivos que começaram a ser desenvolvidos na década de 60 por John Holland na Universidade de Michigan [16]. Este algoritmo é baseado nas teorias da Seleção Natural de Charles Darwin e nos mecan-

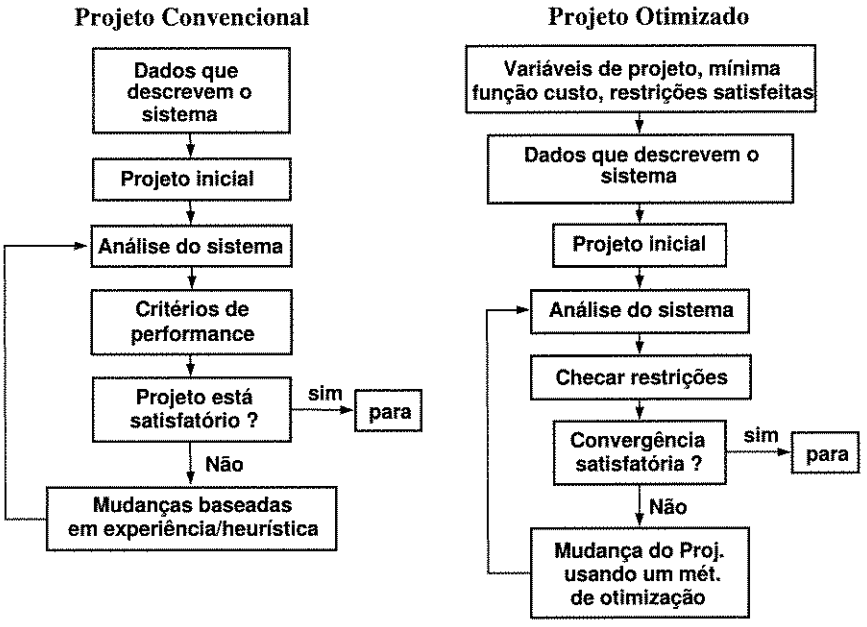


Figura 1.1: Esquema comparativo entre um projeto convencional e um projeto otimizado.

ismos da genética.

Neste trabalho, aplica-se Algoritmo Genético binário ao problema de otimização de estruturas, mais particularmente, problema de otimização topológica de estruturas planas.

O estudo efetuado neste trabalho, engloba três grandes linhas conceituais: a modelagem de estruturas e meios contínuos utilizando técnicas de aproximação numérica, a formulação do problema de otimização topológica e o desenvolvimento e aplicação de um método de busca baseado em Algoritmos Genéticos.

Referindo-se à modelagem de estruturas e meios contínuos, usou-se conceitos básicos de engenharia, sendo que o Método dos Elementos Finitos foi aplicado na obtenção da solução aproximada dos casos estudados ([36], [7], [25], [26]). Foram usados modelos simples, mas representativos, de estruturas treliçadas e meios contínuos bidimensionais governados pelas hipóteses da elasticidade para o caso de estado plano de tensões [6].

A seguir serão apresentados alguns comentários sobre a bibliografia encontrada a respeito das linhas conceituais acima, procurando-se situar ao final, as contribuições realizadas neste trabalho.

1.1 Revisão Bibliográfica

Problemas de otimização estrutural são de grande interesse e estão sendo estudados a algum tempo. A variedade de problemas é muito grande e a maneira com a qual eles são tratados, muito distintas. Esses problemas englobam estruturas com uma, duas ou três dimensões onde se quer minimizar o peso, a compliância, a tensão, flambagem local ou global dentre outros. Muitos métodos já foram aplicados nestes problemas, tais como Programação Linear Quadrática Seqüencial, Método Simplex, Lagrangianos, Máxima descida etc. Dentre os autores que apresentam métodos lineares e não-lineares de otimização estão Bazaraa [2], Luenberger [22], Friedlander [13] entre outros. Alguns estudos específicos no assunto são descritos a seguir:

Vaderplaats [33] faz uma revisão da história da otimização estrutural e identifica o presente estado da arte. Ele faz uma breve introdução nos conceitos de otimização, fala sobre a história da otimização estrutural moderna, apresentando o desenvolvimento dos métodos ao longo dos anos, e finalmente apresenta alguns exemplos.

El-Sayed e Hsiung [10] propõem um novo método para otimização estrutural pela paralelização do Método dos Elementos Finitos. O método consta em dividir a estrutura original em algumas subestruturas e designa cada subestrutura a um processador, e desta forma, cada processador realiza seu cálculo de maneira independente. Com este método os autores visam melhorar o desempenho dos processos de otimização de estruturas, já que o maior esforço computacional desses processos está na análise estrutural. Ao final do artigo alguns exemplos numéricos são apresentados.

Sanat'anna *et al.* [28] apresentam uma aproximação para a otimização estrutural de estruturas reticuladas aeroespaciais 2D e 3D. O objetivo é reduzir o peso das estruturas satisfazendo restrições tais como compliância, tensão nas barras, flambagem local ou global, para uma ou mais cargas diferentes. Começando de uma estrutura básica inicial construída a partir de uma conectividade entre todos os pontos, a melhor topologia é determinada, usando um algoritmo de Programação Linear seqüencial. As variáveis de projeto são as áreas das seções transversais das barras, delimitadas por possíveis valores máximos e mínimos.

A área de otimização estrutural é muito ampla, e não será feita uma revisão bibliográfica aprofundada sobre este tema nesta dissertação. Maiores informações podem ser obtidas nas referências básicas([1], [32] e [15]).

Nas últimas décadas o interesse em resolver problemas baseados em princípios da evolução e hereditariedade vem crescendo. Esses sistemas trabalham com uma população de possíveis soluções, possuem um processo de seleção baseado em um critério de rejeição dado a cada indivíduo e operadores de classificação e seleção. Os

Algoritmos Genéticos são um desses sistemas.

Os Algoritmos Genéticos foram desenvolvidos por Holland [16] na Universidade de Michigan. O seu livro, de 1975, introduziu idéias novas sobre algoritmos de otimização e busca. Os objetivos do algoritmo são abstrair e explicar rigorosamente os processos adaptativos em sistemas naturais, além de desenvolver simulações que retenham os mecanismos originais encontrados em sistemas naturais. Os Algoritmos Genéticos tem como características principais a existência de dois espaços de trabalho: espaço genotípico e espaço fenotípico. O espaço genotípico é a representação codificada de um indivíduo, fazendo uma analogia ao genótipo humano; já o espaço fenotípico representa a estrutura fisicamente, ou seja, o que o realmente se vê. São feitas buscas sobre uma população de pontos e não sobre um único ponto, além de se fazer uso de descrições genéricas do que se quer ver presente na solução, através de funções-objetivo chamadas de funções de *fitness*, utilizando regras de transição probabilísticas e não determinísticas. Posteriormente, outros autores vieram a trabalhar com os Algoritmos Genéticos, entre eles Michalewicz [23] e Goldberg [14]. Em aplicações dos Algoritmos Genéticos em otimização de estruturas, que é a área específica deste trabalho, encontram-se alguns autores descritos a seguir:

Kane *et al.* [17] tratam a otimização estrutural topológica através de Algoritmos Genéticos. O Algoritmo Genético para otimização topológica 2D é testado em uma viga em balanço e em uma placa. O objetivo é otimizar o peso da estrutura sob restrições de deslocamento. A principal vantagem desta aproximação é que pode se encontrar soluções ótimas alternativas e trabalhar com diferentes tipos de modelos mecânicos. Foram feitas algumas adaptações no Algoritmo Genético clássico. O domínio 2D foi discretizado por elementos quadrados que são representados por matrizes, exigindo a criação de novos operadores de *crossover*, chamados de *crossover* diagonal e por blocos, e mutação, chamada de mutação de contorno e epistática. O *fitness* é penalizado e depende fortemente de parâmetros empíricos definido pelo usuário. Alguns resultados em elasticidade com grandes e pequenos deslocamentos são apresentados.

Riche *et al.* [27] consideram o problema de minimização do peso de uma placa de material composto, sujeito a várias restrições. Utiliza-se um Algoritmo Genético com restrições consideradas através de funções de penalidade controladas por parâmetros empíricos. Utiliza-se também um Algoritmo Genético Segregado, isto é, um Algoritmo Genético que divide-se em dois grupos de soluções que tem diferentes níveis de satisfação das restrições, possui estratégia de dupla penalidade e é superelitista, pois este algoritmo é menos sensível à escolha dos parâmetros de penalidade. Neste algoritmo, dois parâmetros de penalidade são usados para cada restrição, e esses valores são associados com dois grupos que têm diferentes níveis de satisfação nas soluções. A estratégia superelitista garante que o melhor indivíduo continue na população.

Alguns resultados comparativos são apresentados.

Yokota *et al.* [35] formulam um problema de minimização do peso de uma treliça de 10 barras com restrições de deslocamento e tensão como um problema estrutural estaticamente indeterminado. A proposta é resolver este problema diretamente, através da utilização de um Algoritmo Genético melhorado. Neste Algoritmo Genético proposto o comprimento das barras e a seção transversal inicial são dados. O cromossomo é formado pela seção transversal de cada barra e a otimização ocorre em torno desses valores. Compara-se a eficiência entre o método proposto e os métodos de critério de otimalidade discretizado.

Deb e Gulati [8] encontram, simultaneamente, a seção transversal, a topologia e a configuração ótimas para treliças 2-D e 3-D com peso mínimo através de Algoritmos Genéticos com código-real. Os genes dos cromossomos utilizados são as seções transversais das barras que compõem a treliça. Como esses genes possuem valores reais, os autores desenvolveram operadores de *crossover* e mutação especiais. Comparando-se os resultados dos testes com os melhores resultados encontrados na literatura observamos um ótimo desempenho do Algoritmo Genético desenvolvido.

Tanimura *et al.* [31] desenvolvem um Algoritmo Genético Distribuído (DGA), isto é, um método de otimização por Algoritmo Genético para computadores em paralelo. Nesse trabalho eles discutem a aplicação do DGA em problemas de otimização de estruturas, ampliando a aplicabilidade do algoritmo proposto a problemas com restrições. Algumas simulações são apresentadas, analisando-se o desempenho do método.

Kawamura *et al.* [18] apresentam um novo procedimento de busca estocástica baseada em Algoritmos Genéticos para desenvolver topologias quase-ótimas para estruturas reticuladas que suportam cargas. Diferentemente dos métodos convencionais, que representam a treliça como uma combinação de membros, eles expressaram a topologia da treliça como uma combinação de triângulos, a fim de obter 100% de estruturas estáveis. Através de alguns exemplos numéricos eles mostram a eficácia e eficiência do método.

Lagaros *et al.* [20] apresentam a investigação de vários algoritmos evolucionários, tais como Algoritmos Genéticos e estratégias de evolução, quando aplicados à problemas de grande-escala em otimização estrutural de tamanho. Ambos os algoritmos imitam evoluções biológicas na natureza e combinam o conceito de sobrevivência artificial do mais apto com operadores evolucionários para formar um mecanismo de busca robusto. Versões modificadas dos algoritmos evolucionários básicos são implementadas para melhorar o desempenho do processo de otimização. Os algoritmos são combinados com métodos de programação matemática para formar metodologias híbridas.

A otimização estrutural utilizando Algoritmo Genético depende diretamente da

análise pelo Método dos Elementos Finitos, pois as estruturas geradas pelo método devem ser avaliadas pela sua estabilidade e pelos deslocamentos nos nós e tensões nas barras. Desta forma, a implementação deste algoritmo deve ser feita da maneira mais concisa possível e que torne o processo mais ágil. Por esses motivos a linguagem C++ orientada a objeto é muito utilizada para essas implementações pois esta é uma linguagem que facilita a manutenção e o desenvolvimento de sistemas computacionais complexos.

A seguir alguns autores que trabalham na linha de otimização estrutural usando Algoritmos Genéticos com implementação usando conceitos de orientação por objetos. Ressalta-se que esta área é bastante ampla e não será revisada em detalhes:

Eldred *et al.* [11] trabalham com um software de design orientado a objeto em C++. O objetivo é fornecer um kit de ferramentas multidisciplinares que seja flexível, extensível e robusto e que estabeleça o protocolo para a interligação da otimização com simulações computacionalmente intensivas. O objetivo é viabilizar o desenvolvimento e a aplicação de algoritmos com estratégias de otimização avançadas, capazes de superar as dificuldades computacionais associadas com códigos de simulação complexos, nas mais diversas áreas.

Krishnamoorthy *et al.* [19] discutem o design orientado a objeto e a implementação de uma biblioteca central de Algoritmo Genético que consiste de todos os operadores genéticos tendo uma interface com uma função objetivo genérica. É mostrado como as classes derivadas dessa biblioteca implementada podem ser usadas para a otimização paramétrica prática de grandes treliças espaciais. Conclui-se que os Algoritmos Genéticos implementados, como o uso de estruturas de dados eficientes e flexíveis, podem ser uma ferramenta muito usada em engenharia de design e otimização.

1.2 Motivação e Descrição da Dissertação

Diante da revisão bibliográfica acima percebe-se que a área de otimização topológica de estruturas é muito ampla e que existe uma infinidade de métodos a serem estudados. Desta forma, é preciso escolher um problema bem definido e um método a ser aplicado na resolução deste problema. Os Algoritmos Genéticos foram escolhidos por sua atualidade e flexibilidade, que despertaram o interesse e a curiosidade necessários para o desenvolvimento da pesquisa.

Os Algoritmos Genéticos podem ser aplicados a qualquer tipo de problema, porém a sua maior vantagem está na resolução de problemas complexos e com um número muito grande de variáveis, onde os métodos tradicionais possuem deficiências. Outras vantagens podem ser apontadas nos Algoritmos Genéticos, como o fato de que eles fornecem um conjunto de possíveis soluções e não uma única solução e

possuem habilidade em sair de ótimos locais.

O trabalho foi dividido em 6 capítulos. No primeiro capítulo situa-se o problema e apresenta-se o atual tratamento do problema e como ele pode vir a ser melhorado. Apresenta-se a proposta de trabalho, a revisão bibliográfica, a motivação para a realização da pesquisa e a organização escolhida para o trabalho.

No capítulo 2 apresenta-se o problema físico a ser resolvido, a descrição matemática do mesmo e a modelagem clássica por Elementos Finitos utilizada. Mostra-se também a formulação do problema de otimização, destacando-se os principais conceitos de otimização topológica.

No terceiro capítulo apresenta-se o Algoritmo Genético, bem como a sua utilização para se resolver o problema de otimização topológica de estruturas.

O capítulo 4 fala sobre a implementação computacional do algoritmo, como ele foi acoplado ao programa *MefLab++* que é desenvolvido por professores e alunos do Departamento de Mecânica Computacional da Faculdade de Engenharia Mecânica da Unicamp e descreve todas as características dessa implementação.

Os resultados numéricos são apresentados no quinto capítulo, juntamente com a análise de desempenho do método.

As conclusões e o trabalhos futuros estão no capítulo 6.

Capítulo 2

Modelagem e Otimização Topológica

A otimização estrutural pode ser aplicada a todos os tipos de estruturas, desde estruturas muito simples como uma barra engastada até estruturas bem mais complexas, como uma estação espacial. Essas estruturas podem ter uma, duas ou três dimensões. A princípio tratam-se apenas de estruturas reticuladas, que por sua simplicidade são mais didáticas e permitem um estudo mais sistemático. Após a validação do algoritmo para o caso de estruturas reticuladas, procura-se estender a abordagem para uma aplicação em estado plano de tensões.

Os procedimentos automáticos de otimização estrutural, pressupõe a existência de um modelo numérico representativo da estrutura em análise. Neste capítulo, mostram-se inicialmente os conceitos básicos da modelagem das estruturas e meios contínuos, revendo-se brevemente os conceitos de discretização e aproximação por Elementos Finitos.

Na seqüência apresenta-se a formulação do problema de otimização topológica, com ênfase no caso de estruturas reticuladas.

2.1 Modelagem de estruturas e meios contínuos

À primeira vista, quando observa-se um problema, vê-se seu aspecto qualitativo, ou seja, enxerga-se a física do problema, seja ela um carro em alta velocidade, um prédio em construção, um avião etc. Neste caso, imagine uma estrutura qualquer. A maioria das pessoas possui uma intuição e experiência bastante razoáveis dos aspectos qualitativos relacionados aos fenômenos mecânicos, o que facilita a sua compreensão. Porém, apenas o conhecimento qualitativo não é suficiente para o bom desenvolvimento de um projeto, pois ele pode levar a vários enganos; logo é preciso ter-se o conhecimento quantitativo do problema [30].

Deve-se desenvolver essa intuição e experiência a fim de conseguir-se uma formulação matemática precisa dos problemas físicos, fazendo-se uma interação entre o problema físico e o problema matemático. A formulação matemática é, ao mesmo tempo, uma linguagem utilizada para descrever os fenômenos físicos e uma ferramenta muito poderosa, pois possibilita o estudo minucioso dos fenômenos. Por esse motivo é muito importante que a formulação matemática descreva o mais corretamente possível o problema em questão.

O método científico para se partir da física e chegar na formulação matemática passa por outros dois passos (Conforme mostrado na Figura 2.1): a observação e experimentação, e a abstração ou indução [24].

1. *Observação e Experimentação*: são o ponto de partida e ao mesmo tempo o teste crucial para a formulação das leis naturais. A física, como ciência natural, parte dos dados experimentais. Mas, é preciso que, após formulada a teoria, seja verificada experimentalmente suas conseqüências, a fim de validar a mesma. Segundo Poincaré:

   "Embora a ciência se construa com dados experimentais, da mesma forma que uma casa se constrói com tijolos, uma coleção de dados experimentais ainda não é ciência, da mesma forma que uma coleção de tijolos não é uma casa."

2. *Abstração, Indução*: o primeiro passo no estudo de um fenômeno natural consiste em fazer abstração de grande número de fatores inessenciais, concentrando a atenção apenas nos aspectos mais importantes. O julgamento sobre o que é ou não importante envolve a formulação de modelos e conceitos teóricos, que representam, segundo Eistein, uma *"livre criação da mente humana"*. A arte está em saber o que e como abstrair, o que é essencial e o que é acessório, aplicando-se as hipóteses corretas.

No caso de estruturas, utiliza-se uma formulação matemática em forma de um sistema de equações diferenciais e condições de contorno.

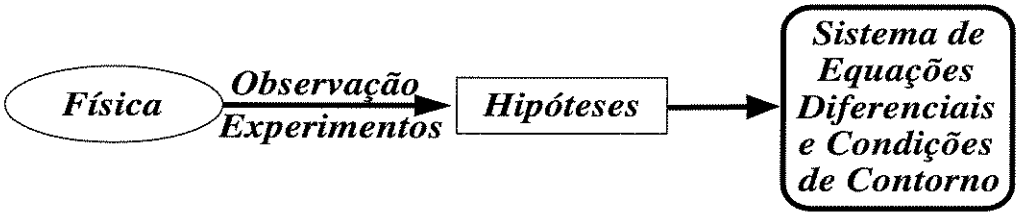


Figura 2.1: Esquema da sistemática para a formulação matemática dos problemas físicos, no caso de estruturas.

O objeto de estudo deste trabalho são estruturas reticuladas e meios contínuos bi-dimensionais. As próximas seções descrevem a modelagem matemática destes dois tipos de estruturas.

2.1.1 Estruturas Reticuladas

2.1.1.1 Descrição e Hipóteses

Estrutura reticulada, ou treliça, é um dos principais tipos de estruturas da engenharia. Ela oferece, ao mesmo tempo, uma solução prática e econômica a muitas situações de engenharia, especialmente no projeto de pontes e edifícios. Uma treliça é formada por barras retas articuladas nas juntas ou nós, como mostra a Figura 2.2. As barras da treliça são interligadas em apenas duas extremidades; assim, nenhuma barra é contínua através de um nó. Como exemplo pode-se observar que a barra AB da Figura 2.2 (a) não existe, pois na verdade ali estão duas barras distintas, AD e DB. Estruturas reais são feitas de várias treliças unidas para formar uma estrutura espacial. Cada treliça é projetada para suportar as cargas que atuam em seu plano e, assim, pode ser tratada como uma estrutura bidimensional [4].

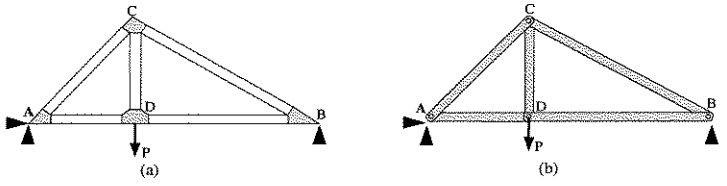


Figura 2.2: Exemplo de uma treliça típica.

Em geral as barras de uma treliça são delgadas e podem suportar pequena carga lateral; todas as cargas, portanto, devem ser aplicadas aos vários nós e não às barras em si. Quando uma carga concentrada é aplicada entre dois nós, ou quando uma carga distribuída é suportada pela treliça, como no caso de uma ponte treliçada, um sistema de piso deve ser previsto, o qual, através do uso de vigas longitudinais e de vigas transversais, transmite a carga aos nós.

Os pesos das barras da treliça são considerados como se estivessem sendo aplicados aos nós, sendo metade do peso de cada barra aplicado a cada um dos dois nós que a barra interliga. Embora as barras sejam, na realidade, unidas por meio de conexões rebitadas ou soldadas, costuma-se considerar que as barras são unidas por meio de pinos; portanto, as forças que atuam em cada extremidade de uma barra reduzem-se a uma única força sem nenhum momento. Assim, as forças consideradas, aplicadas a uma barra de treliça, reduzem-se a uma única força em cada extremidade da barra. Cada barra pode, então, ser tratada como uma barra sob a

ação de duas forças, e a treliça toda pode ser considerada como um grupo de barras com forças aplicadas, como na Figura 2.2 (b). A ação das forças, sobre uma barra individual, pode ocorrer do modo indicado na Figura 2.3. Na primeira, as forças tendem a tracionar a barra, e a barra está sob tração, enquanto na segunda as forças tendem a comprimir a barra, e a mesma está sob compressão.

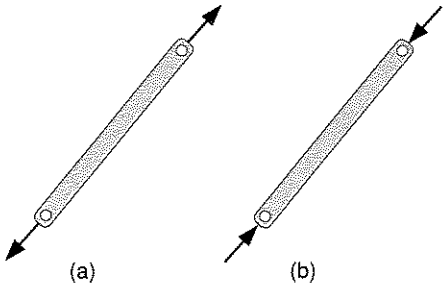


Figura 2.3: (a)Barra sofrendo tração. (b)Barra sofrendo compressão.

Na Figura 2.4 (a) observa-se uma estrutura constituída de quatro barras ligadas por pinos em A, B, C e D. Se uma carga for aplicada em B, a treliça sofrerá grande deformação e perderá completamente sua estabilidade. Por outro lado, a treliça da Figura 2.4 (b), constituída de três barras ligadas por pinos em A, B e C, sofrerá apenas uma leve deformação sob uma carga aplicada em B. Supondo-se que a carga aplicada não cause esforços internos que ultrapassem os limites de escoamento em tração ou compressão nas barras, a única deformação possível para essa treliça é aquela que envolve pequenas alterações no comprimento de suas barras. Tal fato indica que ela não entrará em colapso, isto é, tenderá a assumir uma posição de equilíbrio estável.

Na Figura 2.4 (c), tem-se uma treliça estável maior, que foi obtida pela adição das barras BD e CD à treliça triangular 2.4 (b). Esse procedimento pode ser repetido quantas vezes se desejar, e a treliça resultante será estável internamente se, cada vez que adicionarmos duas novas barras, estas forem ligadas a diferentes nós existentes e interligadas em um novo nó. Esta é uma *treliça simples*.

Deve-se notar que uma treliça simples não é necessariamente composta apenas de triângulos. A treliça da Figura 2.4 (d), é uma treliça simples que foi construída a partir do triângulo ABC, adicionando-se sucessivamente os nós D, E, F e G. Por outro lado, treliças estáveis nem sempre são treliças simples, mesmo quando parecem feitas de triângulos. Um exemplo é a treliça Baltimore (Figura 2.5), pois ela não pode ser construída a partir de um único triângulo.

Observando-se novamente a treliça da Figura 2.4 (b), nota-se que essa treliça possui três barras e três nós. Já a treliça da Figura 2.4 (c) possui duas barras adicionais e um nó adicional, isto é, possui cinco barras e quatro nós. Desta forma,

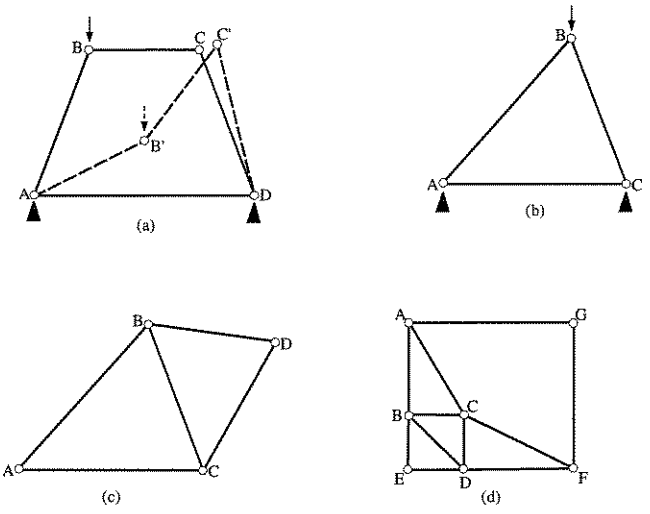


Figura 2.4: (a)Treliça em colapso. (b)Treliça rígida. (c)Treliça Simples. (d)Treliça constituída não só de triângulos.

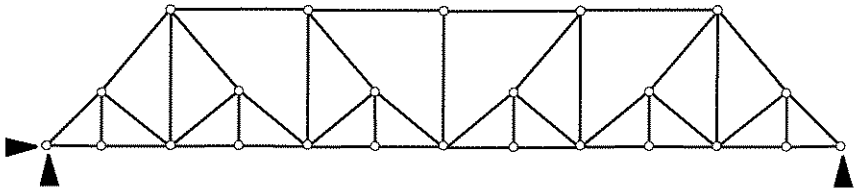


Figura 2.5: Treliça de Baltimore.

cada vez que duas novas barras são acrescentadas, o número de nós aumenta de um, logo, em uma treliça simples, o número total de barras é $b = 2n - 3$, onde n é o número total de nós.

As estruturas treliçadas, bem como alguns problemas de estado plano de tensões foram analisados utilizando-se o Método dos Elementos Finitos.

Neste trabalho a estabilidade estrutural será verificada usando-se as propriedades matemáticas da representação matricial do sistema.

2.1.1.2 Formulação Matemática

Após discutir alguns aspectos físicos do problema e as hipóteses a serem aplicadas, tem-se como próximo passo a formulação matemática do problema, definindo-se as equações diferenciais e as condições de contorno que o governam.

Considerando-se apenas uma barra, como a mostrada na Figura 2.6, sabe-se que a dimensão de seu comprimento é muito maior que as outras dimensões, logo

pode-se reduzir o problema ao caso unidimensional. Para facilitar o entendimento, a obtenção da equação diferencial da barra será dividida em etapas.

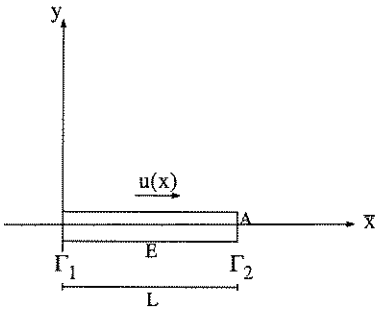


Figura 2.6: Características da barra.

1. Modelo Cinemático Tendo como hipótese a capacidade de se deformar, ou suportar carregamentos ao longo de seu sentido longitudinal, como descrito anteriormente, tem-se que, após a aplicação da carga, todas as seções que eram inicialmente retas, permanecem retas. Deste modo, pode-se fazer as seguintes afirmações em relação à deformação na direção do eixo longitudinal da barra:

O comprimento do elemento diferencial da barra indeformada é Δ_x. Após a aplicação da carga este comprimento é dado por: $\Delta x + (\frac{\partial u}{\partial x}\Delta x)$.

A deformação de uma fibra, a qual é constante em toda a seção, é dada pela relação entre a posição deformada e a posição indeformada:

$$\epsilon_{xx} = \frac{[\Delta x + (\frac{\partial u}{\partial x}\Delta x)] - \Delta x}{\Delta x} \tag{2.1}$$

Desta forma, a deformação da barra é dada, por:

$$\epsilon_{xx} = \frac{du}{dx} \tag{2.2}$$

onde u são os deslocamentos na direção longitudinal da barra.

2. Modelo do Material e Escolha da Lei Constitutiva A partir do tipo do material do qual a barra é constituída define-se uma lei de comportamento, a qual deve relacionar as tensões internas e as deformações da estrutura. Assume-se que o material da barra é homogêneo, isotrópico e tem comportamento linear elástico. Sob estas condições, a *Lei de Hooke* é adotada como lei constitutiva e pode ser escrita como:

$$\sigma_{xx} = E\epsilon_{xx} \tag{2.3}$$

onde, σ_{xx} são as tensões na direção longitudinal da barra, E é o módulo de elasticidade do material e ϵ_{xx} é a deformação na direção longitudinal da barra.

3. Equações de Equilíbrio da Barra Nesta etapa define-se, a partir das *Leis de Newton*, as equações diferenciais de equilíbrio.

Supõe-se que a força de tração à qual a barra da Figura 2.3(a) está sendo submetida tenha intensidade F. Assume-se que a seção reta permanece reta após a deformação e que a distribuição de esforços internos, σ_{xx}, também é constante. Desta forma, aplicando-se a condição de equilíbrio na direção x tem-se:

$$F = \int_A \sigma_{xx} dA = \sigma_{xx} \int_a dA = \sigma_{xx} A \tag{2.4}$$

onde, σ_{xx} é a tensão normal de tração ou compressão e A é a área da seção transversal da barra.

Substituindo-se as equações 2.2 e 2.3 na equação 2.4, tem-se a primeira equação de equilíbrio, dada por:

$$F = \sigma_{xx} A = E\epsilon_{xx} A = EA\frac{\partial u}{\partial x} \tag{2.5}$$

A segunda equação de equilíbrio considera o fato de que a barra esteja submetida a uma carga distribuída de valor $p(x)$. Nestas condições a equação de equilíbrio na direção X é dada por:

$$-F_x + p(x)dx + F_x + dF_x = 0 \tag{2.6}$$

Rearranjando a equação 2.6, tem-se:

$$p(x) = -\frac{dF_x}{dx} \tag{2.7}$$

Substituindo-se a equação 2.4 na equação 2.7 pode-se escrever:

$$p(x) = -\frac{d}{dx}(EA\frac{du}{dx}) \tag{2.8}$$

ou ainda,

$$EA\frac{d^2u}{dx^2} + p(x) = 0 \tag{2.9}$$

Portanto, a equação 2.9 governa o problema analisado e é válida em todo o domínio da barra.

4. Condições de Contorno Para o problema de equilíbrio estático de uma barra deformável, pode-se usar dois tipos de condições de contorno clássicas:

Condições de contorno essenciais: $u = \bar{u}$ em Γ_1, que corresponde à condição de apoio restrito ou de deslocamento imposto.

Condições de contorno naturais: $\frac{\partial u}{\partial x} = \frac{F}{EA}$ em Γ_2, onde $F = 0$ para extremidades livres, e $F = \bar{F}$ nos pontos onde há carga pontual aplicada. Esta condição garante a continuidade das tensões internas com as trações externas nas fronteiras. As cargas distribuídas $p(x)$ encontram-se incluídas na formulação do domínio conforme indicado na equação 2.9.

Desta forma, após desenvolver todas as etapas da formulação matemática, tem-se a equação diferencial e as condições de contorno que regem a barra dadas pela equação 2.10 a seguir:

$$\begin{cases} EA\frac{d^2u}{dx^2} + p(x) = 0 & \text{no domínio } \Omega \\ \text{sujeito a} & \\ u = \bar{u} & \text{em } \Gamma_1 \\ \frac{du}{dx} = \frac{F}{EA} & \text{em } \Gamma_2 \end{cases} \tag{2.10}$$

2.1.2 Meios Contínuos

2.1.2.1 Descrição e Hipóteses

Considere um sólido de espessura uniforme h na direção do eixo z, contornada por dois planos: $z = -\frac{h}{2}$ e $z = \frac{h}{2}$, e um contorno qualquer Γ, conforme mostra a Figura 2.7.

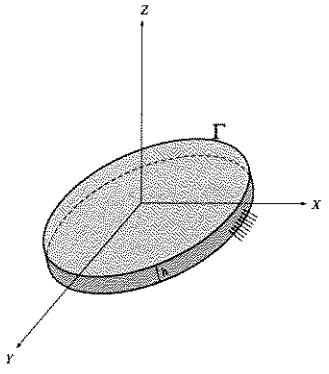


Figura 2.7: Modelo Físico para Problemas de Estado Plano de Tensões.

Quando a espessura h é muito grande, tem-se um problema de *Estado Plano de Deformação*, e quando a espessura h é muito pequena, comparada às dimensões laterais (x e y), tem-se um problema de *Estado Plano de Tensão*. Ambos são simplificações de problemas de elasticidade tridimensional sob as seguintes hipóteses no carregamento:

As forças de volume e as forças aplicadas ao contorno, se existirem, não possuem componentes na direção normal ao plano da estrutura.

As forças de volume não dependem da coordenada da espessura, e as forças aplicadas no contorno são uniformemente distribuídas através da espessura.

Não são aplicados carregamentos nos planos paralelos ao contorno das superfícies do topo e da base.

A hipótese de que as forças são nulas nos planos paralelos implicam no fato das tensões associadas à direção z serem extremamente pequenas para o problema de *Estado Plano de Tensão*, desta forma:

$$\sigma_{zz} = \sigma_{yz} = \sigma_{xz} = 0 \tag{2.11}$$

Já para o problema de *Estado Plano de Deformação*, a hipótese é que as deformações associadas à direção z são nulas, e deste modo, tem-se:

$$\epsilon_{zz} = \epsilon_{yz} = \epsilon_{xz} = 0 \tag{2.12}$$

O campo de tensões na seção transversal de uma viga de seção retangular é um bom exemplo para ilustrar a diferença entre problemas de *Estado Plano de Tensão* e *Estado Plano de Deformação*: se a viga é muito curta, em relação às outras dimensões, considera-se um problema de estado plano de tensão e se a viga é muito longa, considera-se um problema de estado plano de deformação.

2.1.2.2 Formulação Matemática

A análise do *Estado Plano de Tensão* será feita a partir das equações que governam o problema de elasticidade tridimensional, simplificadas através da hipótese descrita na equação 2.11. Como na seção 2.1.1, para facilitar o entendimento, divide-se a formulação em etapas.

1. Modelo Cinemático O modelo cinemático representa as relações entre as deformações e deslocamentos. O modelo cinemático que pode ser adotado neste caso é uma simplificação de um modelo cinemático tridimensional. As seguintes equações definem relações de deformação-deslocamento para o estado plano de tensões [7]:

$$\epsilon_{xx} = \frac{\partial u}{\partial x} \tag{2.13}$$

$$\epsilon_{yy} = \frac{\partial v}{\partial y} \tag{2.14}$$

$$\epsilon_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{2.15}$$

onde u e v são os deslocamentos nas direções x e y respectivamente.

2. Equações Constitutivas do Material Considerando-se o caso de materiais homogêneos, isotrópicos e com comportamento linear elástico, pode-se adotar a *Lei de Hooke* generalizada como lei constitutiva. Para o caso bidimensional, tem-se:

$$\epsilon_{xx} = \frac{1}{E}(\sigma_{xx} - \nu\sigma_{yy}) \tag{2.16}$$

$$\epsilon_{yy} = \frac{1}{E}(\sigma_{yy} - \nu\sigma_{xx}) \tag{2.17}$$

$$\epsilon_{zz} = -\frac{\nu}{E}(\sigma_{xx} + \sigma_{yy}) \tag{2.18}$$

$$\epsilon_{xy} = \frac{1}{G}\sigma_{xy} \tag{2.19}$$

$$\epsilon_{xz} = 0 \tag{2.20}$$

onde, σ_{xx} é a tensão na direção x, E é o *Módulo de Young*, ν é o *Coeficiente de Poisson* e G é o módulo de cisalhamento, sendo:

$$G = \frac{E}{2(1+\nu)} \tag{2.21}$$

Substituindo-se as equações 2.16 e 2.17 na equação 2.18, tem-se:

$$\epsilon_{zz} = \frac{-\nu}{1-\nu}(\epsilon_{xx} + \epsilon_{yy}) \tag{2.22}$$

A equação 2.22 indica que a deformação normal ϵ_{zz} é linearmente dependente das deformações ϵ_{xx} e ϵ_{yy}.

Logo, conclui-se que a deformação não depende de z, mas apenas de x e y. Contudo, encontrou-se uma deformação não nula na direção z, mostrando que o estado plano de tensão não implica em um correspondente estado plano de deformação.

Explicitando-se a *Lei de Hooke* em termos de tensão e reagrupando as constantes nela presentes, tem-se:

$$\sigma_{xx} = c_1\epsilon_{xx} + c_2\epsilon_{yy} \tag{2.23}$$
$$\sigma_{yy} = c_2\epsilon_{xx} + c_1\epsilon_{yy} \tag{2.24}$$
$$\sigma_{xy} = 2c_3\epsilon_{xy} \tag{2.25}$$

onde,

$$c_1 = \frac{E}{1-\nu^2} \tag{2.26}$$
$$c_2 = \frac{\nu E}{1-\nu^2} \tag{2.27}$$
$$c_3 = \frac{E}{2(1+\nu)} \tag{2.28}$$

3. Equações de Equilíbrio em Termos de Tensão Sendo um corpo em repouso ou em movimento retilíneo uniforme, as equações de equilíbrio, em termos de tensões, podem ser escritas da seguinte maneira:

$$\left.\begin{array}{l} \frac{\partial\sigma_{xx}}{\partial_x} + \frac{\partial\sigma_{xy}}{\partial_y} + f_x = 0 \\ \frac{\partial\sigma_{yx}}{\partial_x} + \frac{\partial\sigma_{yy}}{\partial_y} + f_y = 0 \end{array}\right\} \text{em } \Omega \in \Re^2 \tag{2.29}$$

onde, f_x é a componente da força de volume na direção x, f_y é a componente da força de volume na direção y e Ω é um domínio pertencente ao espaço $\Re^2$ [25].

Estas equações são determinadas fazendo-se o equilíbrio de um elemento infinitesimal de lados dx e dy.

4. Condições de Contorno Considerando-se um problema de equilíbrio, pode-se usar dois tipos de condições de contorno clássicas:

Condição de Contorno Natural ou Fórmula de Cauchy:

$$\left.\begin{array}{l} \sigma_{xx}\eta_x + \sigma_{xy}\eta_y = \hat{t}_x \\ \sigma_{xy}\eta_x + \sigma_{yy}\eta_y = \hat{t}_y \end{array}\right\} \text{em } \Gamma_1 \tag{2.30}$$

Condição de Contorno Essencial:

$$\left.\begin{array}{l} u = \hat{u} \\ v = \hat{v} \end{array}\right\} \text{em } \Gamma_2 \tag{2.31}$$

onde, $\hat{\eta} = (\eta_x, \eta_y)$ denota uma normal unitária ao contorno Γ, Γ_1 e Γ_2 são porções do contorno Γ, $\hat{t_x}$ e $\hat{t_y}$ são componentes das forças de superfícies (trações) especificadas no contorno e $\hat{u}$ e $\hat{v}$ são as componentes dos deslocamentos especificados, conforme Figura 2.8.

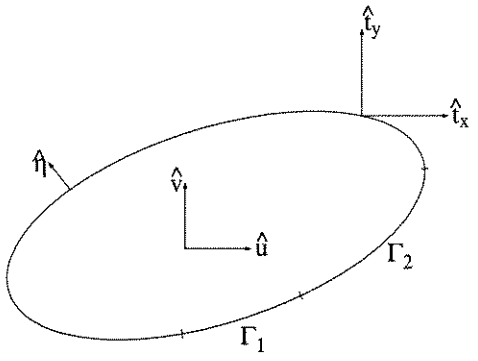


Figura 2.8: Modelo Geométrico e Condições de Contorno.

Desta forma, após desenvolver todas as etapas da formulação matemática, e após algumas manipulações algébricas, tem-se as equação diferenciais e as condições de contorno que descrevem o problema em função dos componentes de tensão, dados por:

Encontrar σ_{xx}, σ_{yy} *e* σ_{xy}, *tal que:*

$$\left.\begin{array}{l} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + f_x = 0 \\ \frac{\partial \sigma_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + f_y = 0 \end{array}\right\} \text{ em } \Omega \in \Re^2$$

sujeito a

$$\left.\begin{array}{l} \sigma_{xx}\eta_x + \sigma_{xy}\eta_y = \hat{t_x} \\ \sigma_{xy}\eta_x + \sigma_{yy}\eta_y = \hat{t_y} \end{array}\right\} \text{ em } \Gamma_1 \qquad (2.32)$$

$$\left.\begin{array}{l} u = \hat{u} \\ v = \hat{v} \end{array}\right\} \text{ em } \Gamma_2$$

De posse dos modelos acima, parte-se para a etapa final, a resolução destes modelos, que será feita pelo Método dos Elementos Finitos.

2.2 O Método dos Elementos Finitos

2.2.1 Introdução

O Método dos Elementos Finitos consiste em uma técnica usada para construir soluções aproximadas para problemas de valor de contorno em equações diferenciais, ordinárias e parciais ([3], [36]).

Na seqüência apresentam-se alguns comentários gerais sobre o Método dos Elementos Finitos, sem no entanto detalhar os fundamentos e a base matemática do mesmo, uma vez que o principal foco deste trabalho é na área de otimização topológica e não na modelagem dos sistemas mecânicos.

Em linhas gerais o método divide o domínio da solução em um número finito de subdomínios simples e usa conceitos variacionais para construir uma aproximação da solução.

As formulações matemáticas mostradas nas seções 2.1.1 e 2.1.2 são problemas de valor de contorno linear, em equações diferenciais onde as variáveis incógnitas são funções que pertencem à classe C^2.

Muitas vezes esta formulação, chamada *formulação clássica* do problema, limita sua resolução pois, em várias aplicações, ou não existe solução por não haver suavidade suficiente dos dados ou, mesmo existindo uma solução suave ela não pode ser encontrada numa forma fechada, devido à complexidade do domínio, dos coeficientes e condições de contorno.

Por isso o problema é reformulado de modo que ele admita condições mais fracas nas soluções e suas derivadas. Uma formulação fraca do problema é dada por: encontre uma função genérica $u(x)$ tal que a equação diferencial governante, juntamente com as condições de contorno, são satisfeitas no sentido de pesos médios. Esta estratégia é conhecida como *formulação fraca* ou *variacional* do problema. Com essa formulação fraca o conjunto X onde procura-se a solução $u(x)$ aumenta.

A especificação de um conjunto apropriado de funções-teste (funções-teste ou funções-peso são funções de x suficientemente bem comportadas tais que são usadas para a ponderação na formulação fraca do problema) é um ingrediente essencial à uma formulação fraca aceitável. A função-teste em problemas variacionais pode não pertencer à mesma classe de funções a que a solução pertence, isto é, pode não pertencer à classe de funções viáveis. Este tipo de formulação não é indicada, pois leva o problema a uma falta de simetria, que pode ser resolvida com a escolha adequada de uma mesma classe para as funções-teste e viáveis, reduzindo assim o esforço computacional.

Ao passar para esta formulação enfraquece-se as exigências de suavidade na solução e, deste modo, progressivamente expande-se a classe de funções para as quais a formulação do problema faz sentido.

O problema em questão é um problema linear e trabalha-se em espaços de dimensão infinita. Utilizando-se aproximações da solução, em espaços de dimensão finita, as funções-teste utilizadas também pertencerão aos mesmos espaços de dimensão finita.

Para encontrar essas soluções aproximadas $\tilde{u}(x)$ utiliza-se o Método de Galerkin, que consiste em, usando a formulação fraca, procurar uma solução aproximada para o problema em um subespaço de dimensão finita dentro do espaço de funções admissíveis.

Como o Método de Galerkin não possui um caminho sistemático de construção das funções de base utilizadas para encontrar as aproximações para a solução, ele dá um desconcertante número de possibilidades e, sabendo que a qualidade da solução aproximada depende muito das propriedades da função de base escolhida, isto passa a ser uma deficiência deste método.

O Método dos Elementos Finitos dá uma técnica geral e sistemática para a construção de funções base para as aproximações de Galerkin dos problemas de valor de contorno. A idéia principal é que uma função base possa ser definida em sub-regiões do domínio, chamadas Elementos Finitos, e que nessas regiões ela possa ser escolhida como uma função muito simples.

Assim, o critério fundamental para a construção dessas funções de base é que as funções sejam geradas por funções simples, definidas por partes sobre a malha de Elementos Finitos, que as funções de base sejam suficientemente suaves para serem membros da classe de funções teste escolhida e que as funções base sejam escolhidas de tal modo que os parâmetros que definem a solução aproximada sejam precisamente os valores de $\tilde{u}(x)$ nos pontos nodais.

Se a base for linear por partes a aproximação por Elementos Finitos para a solução do problema de valor de contorno também será linear por partes, mas seus valores nodais (um nó é um elemento da malha) geralmente não coincidirão com aqueles da solução exata.

2.2.2 Aproximação do Problema de Estruturas Reticuladas

Considere o problema descrito pela equação 2.10. Esta equação rege a barra, isoladamente a qual posteriormente, formará uma treliça. Para garantir-se a suavidade do problema, primeiramente reformula-se a equação diferencial através do *Método dos Resíduos Ponderados*, a fim de tornar a sua resolução menos limitada.

Uma formulação fraca do problema descrito na equação 2.10 é:

Encontre uma função u tal que :

$$\int_\Omega k(\frac{dW_l}{dx}\frac{du}{dx})dx - \int_\Omega W_l p(x)dx - \int_{\Gamma_1} k\frac{du}{dx}W_l d\Gamma - \int_{\Gamma_2} k\frac{du}{dx}W_l d\Gamma = 0 \text{ para todo } W \in H. \tag{2.33}$$

onde W_l são as funções de ponderação ou funções-teste, $k = EA$, Γ_1 e Γ_2 são os contornos e Ω é o domínio.

As funções-teste W são funções de x, suficientemente bem comportadas e tais que sua integral faz sentido. H denota o conjunto das funções suficientemente suaves para serem consideradas funções-teste, e que satisfaçam as condições de contorno do problema.

Essa formulação é conseguida através da minimização do resíduo R, que é uma função em $\Omega = [\Gamma_1, \Gamma_2]$ e é dado por:

$$R_\Omega = L(u) - f \tag{2.34}$$

onde, L é um operador diferencial linear e f independe de u.

Ponderando-se o erro e integrando-se em Ω, obtem-se:

$$\int_\Omega W_l R_\Omega d\Omega = 0 \qquad\qquad l = 1, 2, 3, \ldots, M \tag{2.35}$$

onde W_l é um conjunto de funções de ponderação.

Há dois pontos que devem ser evidenciados. O primeiro é que a formulação mais fraca é tão válida e significativa quanto a formulação original. De fato, a solução da formulação original também satisfaz 2.10 e é sua única solução. O segundo ponto é que a especificação do conjunto H de funções-teste é um ingrediente essencial a uma formulação fraca aceitável.

O próximo passo é a escolha das funções de ponderação, que será baseada no *Método de Galerkin.*

Pelo Método de Galerkin as funções de ponderação são escolhidas como sendo as funções de forma N_l , que são utilizadas na aproximação. Desta forma tem-se: $W_l = N_l$.

Logo, a equação 2.33, na sua forma integral, fica da seguinte forma:

$$\int_\Omega K\Big(\frac{dN_l}{dx}\frac{du}{dx}\Big)dx = \int_\Omega N_l p(x)dx + \left[K\frac{du}{dx}N_l\right]_{\Gamma_1} + \left[K\frac{du}{dx}N_l\right]_{\Gamma_2} \tag{2.36}$$

O Método de Galerkin consiste em procurar uma solução aproximada da formulação variacional simétrica num subespaço de dimensão finita de H , o espaço das funções admissíveis. Mas, o Método de Galerkin possui uma deficiência, pois não há um caminho sistemático de construção de funções-base ou funções de forma razoáveis para a aproximação de funções-teste, conforme observado na introdução.

Existe um número enorme de possibilidades, e é preciso fazer a escolha certa, pois a qualidade da solução aproximada dependerá muito fortemente das propriedades da função de forma que se escolheu. Por isso o Método Clássico de Galerkin é de uso limitado.

O Método dos Elementos Finitos dá uma técnica geral e sistemática para a construção de funções de forma para as aproximações de Galerkin nos problemas de valor de contorno. A idéia principal é que a função de forma possa ser definida em sub-regiões do domínio, chamadas Elementos Finitos e que sobre um sub-domínio, se possa escolhê-la como uma função muito simples, tipo polinômios de grau baixo . Constrói-se então uma malha de Elementos Finitos, que é uma coleção de elementos e pontos nodais fazendo o domínio do problema aproximado.

O critério fundamental para a construção de funções de forma é o seguinte: as funções de forma são geradas por funções simples definidas por partes - elemento por elemento - sobre a malha de Elementos Finitos,são suficientemente suaves para serem membros da classe H de funções-teste e são escolhidas de tal modo que os parâmetros que definem a solução aproximada $\tilde{u}$ sejam precisamente os valores de $\tilde{u}$ nos pontos nodais.

Neste caso, as integrais definidas existentes nas equações de aproximação podem ser obtidas somando-se as contribuições em cada elemento:

$$\int_{\Omega} W_l R_{\Omega} d\Omega = \sum_{e=1}^{E} \int_{\Omega^e} W_l R_{\Omega^e} d\Omega^e \tag{2.37}$$

onde E é o número total de elementos.

A principal vantagem deste procedimento é o fato de tornar possível a aproximação de curvas relativamente complexas, de maneira mais fácil, ao subdividí-las em curvas simples.

A análise do elemento estrutural da barra é um caso no qual apenas um subdomínio é utilizado. Portanto, o domínio $\Omega = [\Gamma_1, \Gamma_2]$ considerado até então corresponde a apenas um elemento, e a fronteira do problema, Γ_1 e Γ_2, corresponde às extremidades da barra, isto é, os seus nós.

A aplicação do Método dos Elementos Finitos induz à obtenção de uma aproximação $\tilde{u}$, válida no domínio $\Omega = 0 \leq \bar{x} \leq L$ do elemento que possui a seguinte forma:

$$u \cong \tilde{u} = \sum_{m=1}^{M} u_m N_m \tag{2.38}$$

onde, os u_m são parâmetros utilizados na aproximação e as N_m são funções de forma.

Os parâmetros u_m e as funções de forma devem garantir uma aproximação de maneira que $\tilde{u} \rightarrow u$ quando $M \rightarrow \infty$. Para um valor finito de M, a equação 2.38 pode ser representada matricialmente por:

$$\tilde{u} = [N]\{u\} \tag{2.39}$$

Desta forma, escolhe-se a função de forma baseada no suporte geométrico adotado. No caso da barra, a variável primária da equação diferencial 2.10, corresponde ao deslocamento nodal u, o qual pode ser aproximado da seguinte maneira:

$$u(\bar{x}) \cong \tilde{u}(\bar{x}) = \left\{ \begin{array}{cc} 1 & \bar{x} \end{array} \right\} \left\{ \begin{array}{c} \alpha_1 \\ \alpha_2 \end{array} \right\} \tag{2.40}$$

onde $\bar{x}$ representa um referencial local do elemento.

Esta aproximação pode ser avaliada em $\bar{x} = 0$ e $\bar{x} = L$, pontos que correspondem aos nós da barra, e, consequentemente à fronteira do problema. Tem-se, portanto:

$$\tilde{u}(0) = \alpha_1 + \alpha_2 0 \quad = \quad \bar{u}_i \tag{2.41}$$

$$\tilde{u}(L) = \alpha_1 + \alpha_2 L \quad = \quad \bar{u}_j \tag{2.42}$$

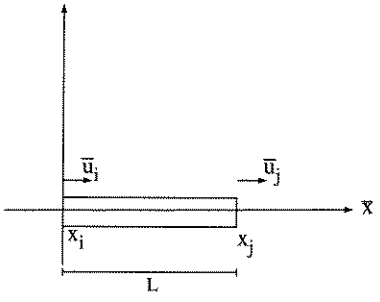


Figura 2.9: Esquema explicativo de uma barra.

onde $\bar{u}_i$ e $\bar{u}_j$ são os deslocamentos nodais do elemento.

Este sistema de equações pode ser representado da seguinte forma:

$$\left\{ \begin{array}{c} \bar{u}_i \\ \bar{u}_j \end{array} \right\} = \left[\begin{array}{cc} 1 & 0 \\ 1 & L \end{array} \right] \left\{ \begin{array}{c} \alpha_1 \\ \alpha_2 \end{array} \right\} \tag{2.43}$$

o que leva a:

$$\left\{ \begin{array}{c} \alpha_1 \\ \alpha_2 \end{array} \right\} = \left[\begin{array}{cc} 1 & 0 \\ \frac{-1}{L} & \frac{1}{L} \end{array} \right] \left\{ \begin{array}{c} \bar{u}_i \\ \bar{u}_j \end{array} \right\} \tag{2.44}$$

Substituindo a equação 2.44 na equação 2.40, chega-se a:

$$\tilde{u}(\bar{x}) = \left\{ \begin{array}{cc} 1 - \frac{\bar{x}}{L} & \frac{\bar{x}}{L} \end{array} \right\} \left\{ \begin{array}{c} \bar{u}_i \\ \bar{u}_j \end{array} \right\} \tag{2.45}$$

Deste modo, tem-se que as funções de forma correspondem a:

$$\begin{aligned} N_i &= 1 - \frac{\bar{x}}{L} \\ N_j &= \frac{\bar{x}}{L} \end{aligned}$$

Substituindo as funções de forma na equação 2.36, que é a forma fraca da equação diferencial que descreve a barra, chega-se à formulação clássica de barras pelo Método dos Elementos Finitos, que é dada por:

$$\frac{EA}{L} \left[\begin{array}{cc} 1 & -1 \\ -1 & 1 \end{array} \right] \left\{ \begin{array}{c} \bar{u}_i \\ \bar{u}_j \end{array} \right\} = \left\{ \begin{array}{c} \bar{F}_i \\ \bar{F}_j \end{array} \right\} \tag{2.46}$$

ou seja,

$$\left[\bar{K}^e\right] \left\{\bar{u}^e\right\} = \left\{\bar{F}^e\right\} \tag{2.47}$$

onde, $\left[\bar{K}^e\right]$ é chamada Matriz de Rigidez de um elemento, $\{\bar{u}^e\}$ é o vetor dos deslocamentos nodais e $\left\{\bar{F}^e\right\}$, o vetor das forças nodais do elemento.

A matriz $[K^e]$ é simétrica, e representa uma relação linear entre esforços e deslocamentos. Neste caso a solução obtida coincide com a solução exata para barras de seção constante submetida a cargas pontuais aplicadas nas extremidades. Caso a geometria do sistema varie, os resultados obtidos com uma formulação deste tipo serão aproximados devido aos erros de geometria e modelagem.

Deve-se ter em mente que a barra, pode ser descrita através de dois sistemas de coordenadas: o sistema local, convenientemente orientado com respeito à barra e o sistema global, convenientemente orientado com respeito à toda estrutura.

Para realizar estas rotações na matriz de rigidez são desenvolvidas matrizes de transformação.

Como a treliça é uma associação de barras, é necessário introduzir ambas as coordenadas: locais e globais. As coordenadas locais são escolhidas para representar uma barra isoladamente; já as coordenadas globais, representam a estrutura completa.

A transformação da matriz de rigidez, de um conjunto de coordenadas para outro, é imprescindível para a resolução da treliça.

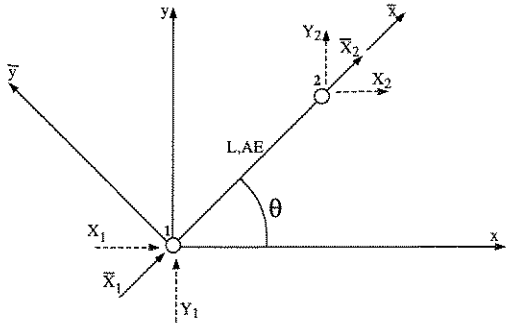


Figura 2.10: Sistema de referência para um elemento de barra.

Observando-se a Figura 2.10, chama-se de $\overline{X}_1$ e $\overline{X}_2$ as coordenadas horizontais locais, e de $\overline{Y}_1$ e $\overline{Y}_2$ as coordenadas verticais locais. Como não se tem forças verticais ($F_{\overline{Y}_1}$ e $F_{\overline{Y}_2}$) no nosso problema, elas são introduzidas de forma a não alterar o sistema:

$$\{\bar{F}\} = \begin{Bmatrix} F_{\overline{X}_1} \\ F_{\overline{Y}_1} \\ F_{\overline{X}_2} \\ F_{\overline{Y}_2} \end{Bmatrix} = \frac{AE}{L} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \overline{u}_1 \\ \overline{v}_1 \\ \overline{u}_2 \\ \overline{v}_2 \end{Bmatrix} = \left[\bar{K}\right] \{\bar{u}\} \tag{2.48}$$

Para construir a matriz de transformação deve-se saber que:

$$\overline{X} = X \cos\Theta + Y \sin\Theta \tag{2.49}$$

$$\overline{Y} = -X \sin\Theta + Y \cos\Theta \tag{2.50}$$

Fazendo-se $\lambda = \cos\Theta$ e $\mu = \sin\Theta$, pode-se escrever a transformação de coordenadas na seguinte forma matricial:

$$\underbrace{\begin{Bmatrix} \overline{X}_1 \\ \overline{Y}_1 \\ \overline{X}_2 \\ \overline{Y}_2 \end{Bmatrix}}_{\{\overline{X}\}} = \underbrace{\begin{bmatrix} \lambda & \mu & 0 & 0 \\ -\mu & \lambda & 0 & 0 \\ 0 & 0 & \lambda & \mu \\ 0 & 0 & -\mu & \lambda \end{bmatrix}}_{[T]} \underbrace{\begin{Bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \end{Bmatrix}}_{\{X\}} \tag{2.51}$$

A matriz de transformação $[T]$ é ortogonal, ou seja, $[T]^{-1} = [T]^t$, e portanto temos que $\{X\} = [T]^{-1}\{\overline{X}\} = [T]^t\{\overline{X}\}$. Passando a matriz de rigidez para as coordenadas globais tem-se $[T]\{F\} = \left[\overline{K}\right][T]\{u\} \Rightarrow \{F\} = [T]^t\left[\overline{K}\right][T]\{u\} =$

$[K]\{u\}$. Agora, utilizando $[\overline{K}]$ e a matriz de transformação definida no sistema de equações (2.51) encontra-se $[K]$ da relação anterior, que é dada por:

$$[K] = \frac{AE}{L}\begin{bmatrix} \lambda^2 & \lambda\mu & -\lambda^2 & -\lambda\mu \\ \lambda\mu & \mu^2 & -\lambda\mu & \mu^2 \\ -\lambda^2 & -\lambda\mu & \lambda^2 & \lambda\mu \\ -\lambda\mu & \mu^2 & \lambda\mu & \mu^2 \end{bmatrix} \tag{2.52}$$

para cada elemento. Esta é a matriz de rigidez para a barra da Figura 2.10 referente às coordenadas x e y globais.

Para representarmos barras agrupadas, como é o caso de treliças, vamos sobrepor as matrizes $[K]$ encontradas para cada elemento, obtendo-se uma matriz de rigidez global de dimensão igual a $2n$ onde n é o número de nós da treliça. Para montar esta matriz final do Método dos Elementos Finitos, usa-se o procedimento clássico de montagem do Método dos Deslocamentos, que segue de acordo com a expressão (2.37).

2.2.3 Aproximação do Problema de Elasticidade Bidimensional

Considere o problema de estado plano de tensões descrito pela equação 2.32. Primeiramente determina-se a *Formulação Fraca* do problema, utilizando a técnica dos *Resíduos Ponderados.* Alguns detalhes da formulação serão omitidos, pois seguem a mesma idéia da formulação apresentada na seção 2.2.2 [25].

Desta forma, tem-se que a condição de equilíbrio para o problema 2.32 é dada por:

$$\int_\Omega (\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y})\psi(x,y)dxdy = -\int_\Omega f_x \psi(x,y)dxdy \tag{2.53}$$

$$\int_\Omega (\frac{\partial \sigma_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y})\psi(x,y)dxdy = -\int_\Omega f_y \psi(x,y)dxdy \tag{2.54}$$

onde $\psi(x,y)$ são as funções de ponderação que inicialmente não estão sujeitas a nenhum tipo de restrição.

Aplicando-se integração por partes na equação (2.53), a fim de reduzir a ordem das derivadas exigidas sobre σ_{ij}, e após manipulações algébricas, a forma fraca expressa em tensão, sabendo-se que $\psi(x,y) = 0$ em Γ_2 é dada por:

Achar σ_{xx}, σ_{xy}, σ_{yy} *tal que:*

$$\begin{cases} \int_\Omega (\frac{\partial \psi(x,y)}{\partial x}\sigma_{xx} + \int_\Omega (\frac{\partial \psi(x,y)}{\partial y}\sigma_{xy}) dxdy = \int_\Omega f_x \psi(x,y) dxdy + \oint_\Gamma \psi(x,y)\hat{t}_x d\Gamma \\ \int_\Omega (\frac{\partial \psi(x,y)}{\partial x}\sigma_{yx} + \int_\Omega (\frac{\partial \psi(x,y)}{\partial y}\sigma_{yy}) dxdy = \int_\Omega f_y \psi(x,y) dxdy + \oint_\Gamma \psi(x,y)\hat{t}_y d\Gamma \\ \\ \text{s.a} \\ \left.\begin{array}{l} u = \hat{u} \\ v = \hat{v} \end{array}\right\} \text{em} \Gamma_2 \end{cases} \tag{2.55}$$

Utilizando-se o *Método de Galerkin*, adota-se funções ponderadoras ψ como funções de interpolação $N_i(x,y)$.

Na forma clássica de interpolação polinomial nodal, as funções de interpolação $N_i(x,y)$ são aplicadas em cada elemento ou sub-domínio e satisfazem a condição de ter valor unitário no nó i e se anular nos demais nós do elemento. Assim, temos:

$$N_i(x_j, y_j) = \begin{cases} 0, & \text{se } i \neq j \\ 1, & \text{se } i = j \end{cases} \tag{2.56}$$

Isto significa que os valores dos deslocamentos u e v nos nós i representam exatamente os valores numéricos dos deslocamentos u_i e v_i, o que caracteriza a aproximação do tipo nodal [29].

A fim de simplificar as operações necessárias, define-se um elemento de referência isoparamétrico, que é definido em um espaço dimensional abstrato, conforme Figura 2.11. Este elemento permite que, baseado em coordenadas locais, sejam calculadas as grandezas de cada elemento real usando apenas um único conjunto de funções de interpolação.

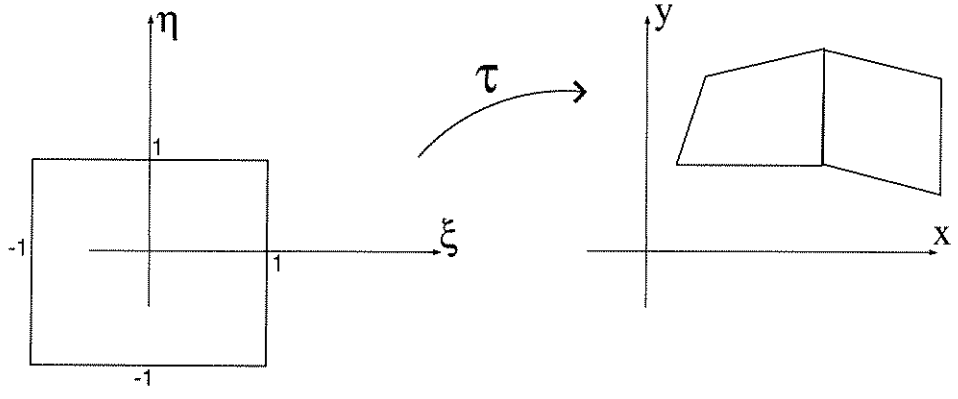


Figura 2.11: Elemento de referência isoparamétrico e elemento real.

A geometria do elemento de referência é mapeada na geometria do elemento real usando expressões de transformação geométrica. No caso do elemento isoparamétrico, as funções de transformação geométrica $\bar{N}_i$ são as mesmas que as funções de aproximação N_i.

Usando o procedimento padrão desenvolvido para o elemento de barra, tem-se as seguintes funções de forma:

$$\begin{aligned} N_1 &= \tfrac{1}{4}(1-\xi)(1-\eta) \\ N_2 &= \tfrac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \tfrac{1}{4}(1+\xi)(1+\eta) \\ N_4 &= \tfrac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{2.57}$$

Substituindo as funções de forma na equação 2.55, que é a forma fraca da equação diferencial que descreve os meios contínuos e, após manipulações algébricas, chega-se à formulação clássica pelo Método dos Elementos Finitos, que é dada por:

$$[K] = \int_{-1}^{1}\int_{-1}^{1} [B(\xi,\eta)]^t [D] [B(\xi,\eta)] \, det(J) d\xi d\eta \tag{2.58}$$

onde, $[B(\xi,\eta)]$ é a matriz de derivadas parciais das funções de interpolação com respeito ao referencial global, $[D]$ é a matriz de propriedades do material e J é o Jacobiano da transformação geométrica.

2.3 O Problema de Otimização Topológica de Estruturas

O problema de otimização topológica de estruturas pode ser abordado de diversas formas. As principais abordagens aplicadas ao problema de treliças referem-se à determinação da ausência ou presença de barras, remoção de nós, variação na posição dos nós, variação na área da seção transversal das barras, variação no comprimento das barras e combinações dos itens acima. As principais categorias de métodos clássicos que se dispõem a resolver esses problemas são três:

- *Otimização Paramétrica:* as áreas das seções transversais são variáveis e as coordenadas dos nós e as conectividades são fixas.
- *Otimização de Forma:* as coordenadas dos nós são variáveis.
- *Otimização de Topologia:* as conectividades são variáveis.

Bons resultados podem ser obtidos pela combinação dos três tipos de abordagens. O problema da minimização da massa total da estrutura [8] é dado por :

$$
\begin{aligned}
minf(A) &= \sum_{j=1}^{m} \rho_j l_j A_j && (2.59)\\
\text{s.a} & \\
G1 &= \text{treliça é aceita.} && (2.60)\\
G2 &= \text{treliça é cinematicamente estável} && (2.61)\\
G3 &= \delta_k^{max} - \delta_k(A,\xi) = 0, k = 1,2,...,n && (2.62)\\
G4 &= S_j - \sigma_j(A,\xi) = 0, j = 1,2,...,m && (2.63)
\end{aligned}
$$

onde, ρ é a densidade do material, l é o comprimento da barra, A é a área da barra, S é a tensão admissível na barra, σ é a tensão normal atuando na barra, ξ representa genericamente as coordenadas dos nós e δ_k é o limite máximo admissível para os deslocamentos de qualquer nó da estrutura.

As varáveis δ_k e σ_j deverão ser obtidas a partir de uma simulação de Elementos Finitos; todas as outras farão parte do arquivo do problema.

Capítulo 3

Otimização Topológica Usando Algoritmos Genéticos

3.1 Introdução

A otimização é considerada hoje uma ferramenta indispensável para a análise de muitos problemas de decisão e alocação. Usando a otimização pode-se maximizar ou minimizar um problema de forma clara e objetiva, melhorando consideravelmente o desempenho de muitos processos.

Um problema de otimização caracteriza-se por uma função objetivo, que descreve o que deseja-se otimizar, isto é, que mede a qualidade da decisão a ser tomada, e restrições, que limitam o espaço de busca da solução, restringindo esta a um grupo de soluções que atendam a certas necessidades.

Os problemas a serem abordados podem ser descritos por funções lineares e não lineares, com ou sem restrições. A formulação matemática do problema é uma etapa muito importante para se conseguir uma boa solução. Se a descrição dos aspectos relevantes do problema (objetivos a serem otimizados e restrições a serem atendidas) for bem feita, a chance de encontrar-se uma boa solução é maior.

Alguns problemas permitem a obtenção de soluções analíticas, isto é, soluções que podem ser expressas algebricamente de forma fechada. Mas a maior parte dos problemas apresentam características que não nos permite obter a solução analítica. Por causa disso foram desenvolvidos métodos iterativos (como Newton), que a partir de uma solução inicial caminham em direção ao ponto ótimo do problema. Os métodos iterativos não garantem a solução ótima do problema. No entanto, é preciso que o método tenha convergência garantida em tempo finito e compatível com as necessidades da aplicação e que garanta a convergência para uma boa solução.

Quando o problema exibe um grau de complexidade elevado, apresentando múltiplos objetivos devidamente ponderados, múltiplas restrições e um número muito

grande de variáveis, levam a uma explosão combinatória de candidatos à solução, representando procedimentos considerados intratáveis caso não se abra mão da solução ótima.

É neste momento que entram em cena as técnicas da computação evolutiva, que se destacam no tratamento destes problemas. Estas técnicas trabalham com a obtenção de vários candidatos à solução ao mesmo tempo, privilegiam as melhores possíveis soluções na obtenção das soluções candidatas seguintes. São capazes de detectar regiões promissoras no espaço de busca e atingir soluções de boa qualidade sem a necessidade de avaliar um grande número de candidatos.

Escolheu-se o Algoritmo Genético para ser aplicado à otimização de estruturas devido à sua generalidade e ao bom desempenho médio do algoritmo apresentado nos trabalhos na área. Neste capítulo será apresentado um breve histórico do algoritmo escolhido e seus princípios de funcionamento, mostrando, de modo geral, as possíveis variações do método. A definição do tipo de algoritmo escolhido e a codificação do problema de otimização de estruturas são os assuntos finais deste capítulo.

3.2 O Algoritmo Genético: Princípio de Funcionamento

Diante da necessidade de se criar novas abordagens a fim de superar as dificuldades ainda existentes para se resolver certos problemas, surgiu a computação natural. Inspirada em fenômenos naturais, ela tem o papel de ser modelo computacional de processos evolutivos naturais e de ser ferramenta de otimização para a solução de problemas.

Nos últimos tempos muitas técnicas inspiradas na natureza vem sendo desenvolvidas, dentre elas estão as redes neurais, lógica nebulosa e computação evolutiva (Ver Figura 3.1). Um dos fatores que desperta o interesse do homem no desenvolvimento de processos de imitação da natureza é a simples comparação de dispositivos naturais e artificiais.

Algoritmos evolutivos são procedimentos computacionais para a solução de problemas ou modelagem de processos evolutivos, resultantes da aplicação de técnicas heurísticas baseadas na seguinte seqüência básica comum: realização de reprodução, imposição de variações aleatórias, promoção de competição e execução de seleção de indivíduos de uma dada população. O fluxograma da Figura 3.2 dá uma idéia geral do procedimento adotado por um Algoritmo Evolutivo.

Originalmente o objetivo dos pesquisadores era o desenvolvimento de um algoritmo que possibilitasse simulações computacionais de sistemas genéticos naturais, através da análise do processo de seleção natural, onde uma função representava o

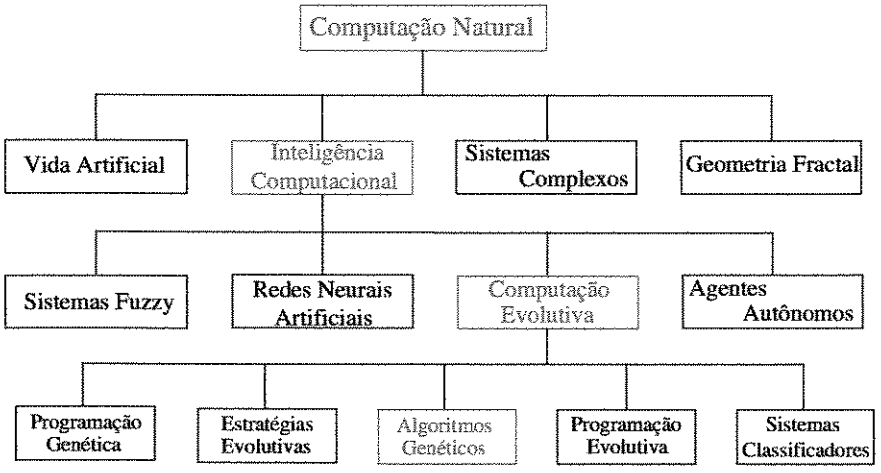


Figura 3.1: Taxionomia dos Sistemas Computacionais Naturais.

grau de adaptabilidade dos indivíduos ao meio em que vivem.

Holland foi o primeiro a perceber a semelhança entre essas simulações e a otimização de funções, sendo também o primeiro a utilizar técnicas similares aos processos genéticos para este propósito. Mas foi em 1967 que esse algoritmo ganhou o nome de algoritmo genético, em um trabalho de Bagley. A partir daí esses algoritmos vem sendo amplamente estudados e aplicados nas mais diversas áreas [5].

Foi somente em 1968, com a *"Teoria dos Esquemas"* que Holland desenvolveu o Teorema Fundamental dos Algoritmos Genéticos, o qual fornece bases mais teóricas para a análise de convergência e desenvolvimento dos algoritmos. A partir daí foram desenvolvidos vários trabalhos de aplicações de desenvolvimento do algoritmo, e foram criados operadores genéticos mais sofisticados.

Em 1975, Holland publicou o livro " Adaptação em Sistemas Naturais e Artificiais", que é o livro mais importante sobre o assunto [16].

Os Algoritmos Genéticos são baseados nas teorias da Seleção Natural de Charles Darwin e nos mecanismos da genética. Eles se utilizam de codificação genética, reprodução, cruzamento entre indivíduos, mutação de genes, maiores chances de sobrevivência do mais adaptado, etc e também de uma população de indivíduos que representam as possíveis soluções. É entre esses indivíduos (soluções)que os processos adaptativos ocorrem. Por possuírem um conjunto de soluções ao invés de uma única solução, os Algoritmos Genéticos são considerados algoritmos de otimização global, pois os riscos de obtenção de ótimos locais são reduzidos.

Para o bom funcionamento do algoritmo, primeiramente é preciso decidir como um indivíduo será codificado. As possíveis soluções são representadas por códigos no algoritmo, o que chamamos de genótipo. Analogamente ao tratamento dado na

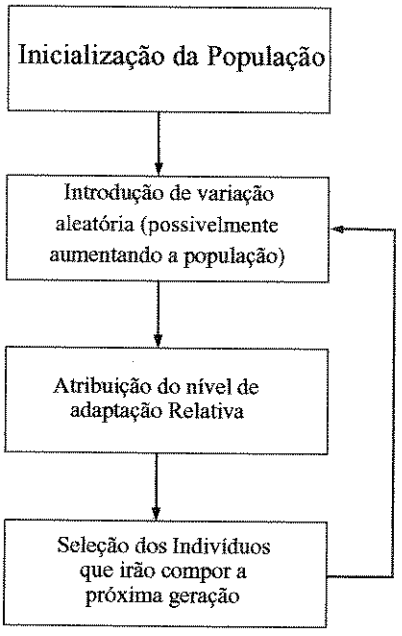


Figura 3.2: Fluxograma de um Algoritmo Evolutivo.

biologia, temos o fenótipo e o genótipo da solução.

Decidida a representação dos indivíduos partimos para a aplicação do algoritmo. Basicamente o algoritmo obtém uma população inicial de indivíduos e evolui essa população ao longo das gerações. Com o decorrer das gerações obteremos populações cada vez mais evoluídas e adaptadas ao meio, e essa evolução se dá através da aplicação de operadores genéticos.

A população inicial é gerada aleatoriamente e o critério de parada é o número de gerações. Os operadores genéticos são: mutação, crossover e seleção, todos baseados no que acontece com os cromossomos.

Existem vários tipos de algoritmos genéticos [34]. Esses tipos são determinados de acordo com os operadores genéticos utilizados. Quanto à codificação podemos ter algoritmos binários (onde os genes são representados por zeros e uns - Figura 3.3), reais (genes são números reais), inteiros (genes são números inteiros) ou mistos (uma combinação de vários tipos de genes). Dependendo da codificação do indivíduos adaptamos os operadores de mutação e crossover.

1	0	0	1	1	1	0	0	0	0	1	0

Figura 3.3: Cromossomo binário.

O crossover pode ser uniforme, simples, duplo etc. O crossover uniforme possui um número variável de pontos de cruzamento. O que se determina é a porcentagem de genes que serão trocados. O crossover simples (Figura 3.4) possui apenas um ponto de cruzamento escolhido aleatoriamente, enquanto o duplo possui dois e assim sucessivamente.

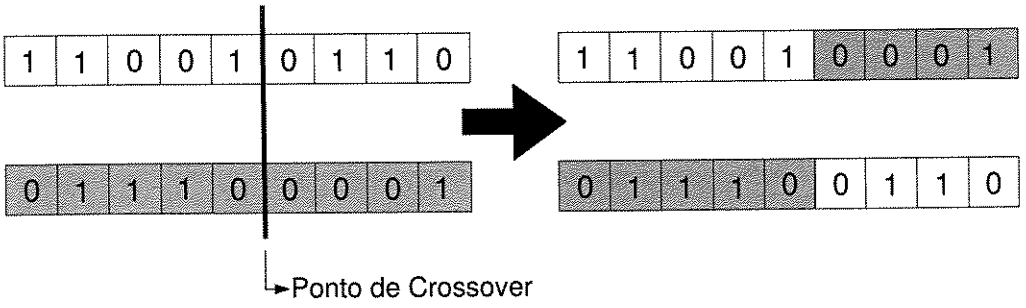


Figura 3.4: Representação esquemática de um crossover simples

Na mutação tem-se duas variantes: a mutação aleatória e a indutiva. A mutação aleatória consiste em determinar a porcentagem de genes que serão trocados e através de um sorteio esses genes são aleatoriamente escolhidos (Figura 3.5). Já a mutação indutiva só é aplicável à codificações reais, é semelhante à anterior, só que sorteia-se um valor a ser somado ao valor corrente do alelo.

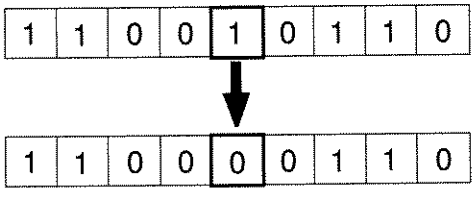


Figura 3.5: Representação esquemática de uma mutacao aleatória

A seleção é feita por Roulette Wheel (Figura 3.6), que consiste na seleção baseada em pesos. Cada indivíduo possui um peso relativo ao valor de seu fitness: quanto melhor o fitness maior o peso. Assim, quanto maior o peso do indivíduo maiores são as chances de ele ser o escolhido e passar suas características adiante.

Existem vários mecanismos de seleção para a criação da nova população. A seleção pode ser elitista, onde garantimos que os melhores indivíduos estarão na próxima geração. Pode ser salvacionista, onde apenas o melhor será garantido. Abaixo estão descritos alguns dos mecanismos de seleção existentes:

Cromossomo	Fitness	Graus
1	6.0	180
2	3.0	90
3	1.5	45
4	1.5	45

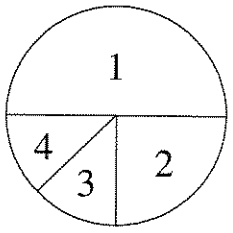


Figura 3.6: Representação de um Roulette Wheel

- **Seleção Elitista**: São selecionados os n melhores indivíduos da população intermediária.
- **Seleção Aleatória**: São selecionados aleatoriamente n indivíduos da população intermediária. A seleção aleatória pode ser:
  - *Salvacionista*: Mantém-se o melhor indivíduo e seleciona-se os outros $n-1$ aleatoriamente
  - *Não-Salvacionista*: Seleciona-se aleatoriamente todos os indivíduos. Neste caso a seleção pode obedecer uma distribuição uniforme, onde todos têm a mesma chance de ser selecionado ou uma distribuição proporcional.
- **Seleção por Diversidade**: São selecionados os indivíduos mais diversos na população intermediária, a partir do melhor indivíduo.
- **Seleção Bi-classista**: São selecionados os $p\%$ melhores indivíduos e os $(100-p)\%$ piores indivíduos.
- **Seleção por Torneio**: Dois indivíduos são escolhidos aleatoriamente. Um número aleatório r entre 0 e 1 é gerado. Se $r < k$(parâmetro) o melhor dos indivíduos é escolhido. Caso contrário escolhe-se o outro.
- **Seleção Steady-State**: A população original é mantida, com a exceção de alguns poucos indivíduos menos adaptados.

Precisa-se também decidir qual será o tipo de abordagem escolhida para o tratamento do problema. Pode-se adotar a abordagem Michigan, onde a população como um todo é a solução para o problema, ou a abordagem Pittsburgh, onde cada elemento da população corresponde a uma solução do problema.

Diante de todas essas opções, escolheu-se trabalhar com um algoritmo genético binário, com crossover uniforme e mutação simples, seleção elitista e abordagem

Pittsburg. Essa configuração foi escolhida após uma análise do problema. A representação binária do cromossomo foi escolhida por entender-se que é a que melhor representa um cromossomo no caso de otimização topológica, pois como neste tipo de otimização os nós são fixos e as seções transversais da barra também, esta representação é suficiente. A escolha do crossover e da seleção se deu após alguns testes e a abordagem Pittsburg se encaixa perfeitamente no problema, pois temos um conjunto de possíveis soluções para o problema, onde cada indivíduo representa uma estrutura. A Figura 3.7, mostra o fluxograma do Algoritmo Genético usado.

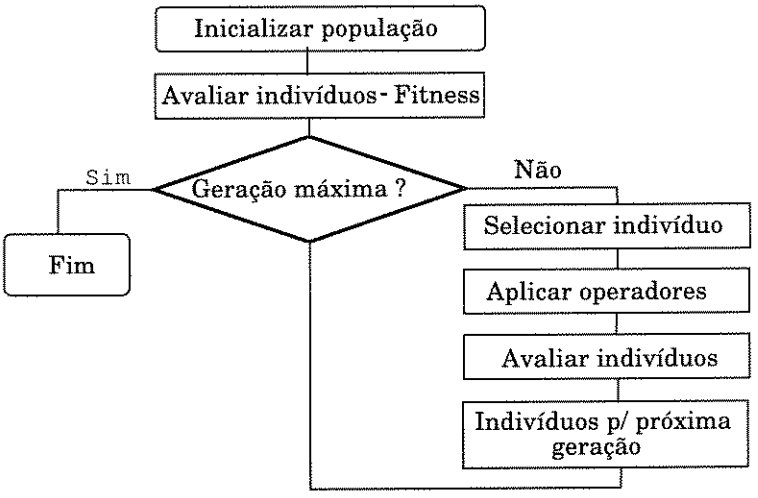


Figura 3.7: Fluxograma do Algoritmo Genético.

3.3 O Algoritmo Genético aplicado à otimização estrutural

O problema a ser resolvido foi apresentado na seção 2.3, descrito pelo seguinte sistema de equações:

$$
\begin{aligned}
minf(A) &= \sum_{j=1}^{m} \rho_j l_j A_j \\
\text{s.a} & \\
G1 &= \text{treliça é aceita.} \\
G2 &= \text{treliça é cinematicamente estável} \\
G3 &= \delta_k^{max} - \delta_k(A,\xi) = 0, k = 1,2,...,n \\
G4 &= S_j - \sigma_j(A,\xi) = 0, j = 1,2,...,m
\end{aligned}
$$

onde, ρ é a densidade do material, l é o comprimento da barra, A é a área da barra, S é a tensão admissível na barra, σ é a tensão normal atuando na barra, ξ representa genericamente as coordenadas dos nós não-básicos e δ_k é o deslocamento nodal do k-ésimo nó, sendo δ_k^{max} o limite máximo admissível para os deslocamentos de qualquer nó da estrutura.

As estruturas tratadas são compostas por elementos e nós, que serão divididos em nós básicos e não-básicos (veja exemplo na F igura 3.8). Os nós não-básicos são opcionais, podendo ser retirados da estrutura. Os nós básicos, por outro lado, são essenciais e devem existir em todas as estruturas resultantes da otimização já que são eles que permitem a aplicação das condições de contorno do problema. Essas informações são especificadas pelo usuário e devem ser respeitadas, podendo seu descumprimento gerar penalidades. O objetivo da otimização é descobrir quais nós opcionais devem pertencer à estrutura assim como os elementos que devem estar presentes na estrutura final a fim de que o peso da estrutura seja o menor possível, satisfazendo certas restrições. A estrutura inicial a ser trabalhada consiste em uma estrutura completa, ou seja, com todos os nós e elementos possíveis.

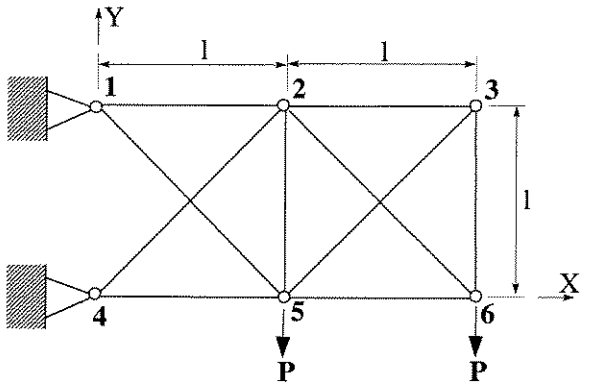


Figura 3.8: Exemplo: Treliça com 6 nós e 10 barras. Os nós básicos são 1, 4, 5 e 6, pois as forças P são aplicadas em 5 e 6 e os nós 1 e 4 estão engastados.

Para a resolução do problema de otimização topológica de estruturas desenvolveu-se um Algoritmo Genético clássico, onde os cromossomos são binários, representando a ausência(0) ou presença(1) de material em uma certa posição da estrutura. Para o caso das treliças, modeladas pelo Método dos Elementos Finitos, isso corresponde à ausência(0) ou presença(1) do elemento de barra em uma certa posição da malha (Figura 3.9). A posição dos nós e a área da seção transversal da barra são dados constantes. O algoritmo realiza crossover e mutação clássicos e dispõe de seleção elitista, mantendo sempre os melhores indivíduos na população. Trabalha-se com

penalizações para estabelecer a factibilidade ou não de uma solução. Primeiramente verifica-se a presença obrigatória dos nós básicos, que são aqueles nós onde são aplicadas as forças ou as restrições; se algum dos nós básicos não estiver presente na configuração da treliça, penaliza-se o valor do fitness e não se verifica mais nada. Mas, se todos os nós básicos estiverem presentes na configuração, passa-se para a próxima etapa. Na seqüência verifica-se se a treliça em questão é estaticamente estável. Para isso, monta-se a matriz de rigidez da mesma e verifica-se se esta é definida positiva ou não. Como no caso anterior, se a condição não for satisfeita penaliza-se o fitness e termina-se com os cálculos. Caso contrário, dá-se procedimento à terceira e última etapa. Nesta etapa, calcula-se o deslocamento nos nós e a tensão nas barras da treliça e depois compara-se esses valores à um valor limite. Penaliza-se o fitness proporcionalmente à soma do módulo das diferenças entre o deslocamento encontrado e o permitido e entre a tensão encontrada e a permitida. Após a realização destes cálculos para todos os elementos da população aplica-se crossover e mutação, criando uma população intermediária maior que a população atual, e inicia-se a construção da população da geração seguinte através da escolha de indivíduos dessa população intermediária.

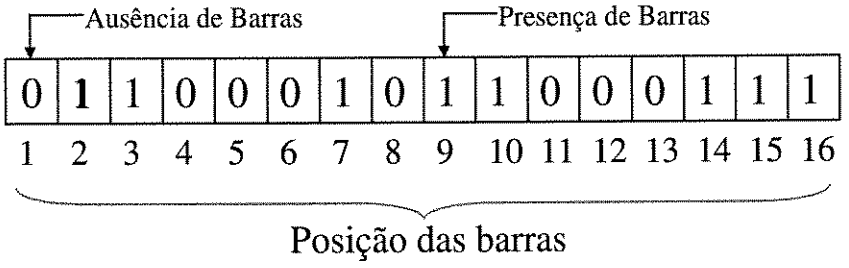


Figura 3.9: Esquematização de um cromossomo binário

As etapas mencionadas acima, serão descritas mais detalhadamente na seção 3.3.2.

3.3.1 Codificação do Problema e Operadores Genéticos

Para esse tipo de problema, a função objetivo, como visto, visa minimizar o peso da estrutura, ou seja, minimizar o número de elementos presentes na estrutura. Outros critérios também poderiam ter sido adotados. É preciso lembrar que todas as restrições devem ser satisfeitas e que cada uma delas gera uma penalidade com valor a ser acrescido na função objetivo. A função de *fitness* para este problema será então dividida em partes, já que cada restrição penaliza a função em um valor. O algoritmo 1 mostra como foram aplicadas as penalidades segundo as restrições.

Algorithm 1 Aplicação das penalidades.

```
1: if restrição 1 for satisfeita then
2:    if restrição 2 for satisfeita then
3:       Fitness = valor função obj. + penalid. restrição 3 + penalid. restrição 4;
4:    end if
5:    Fitness = valor função obj. + penalid. restrição 2;
6: end if
7: Fitness = valor função obj. + penalidade restrição 1.;
```

Ou seja, não se penaliza todas as restrições de uma única vez. Examina-se cada restrição a seu tempo e então aplica-se penalidades caso aquela restrição não tenha sido atendida. Caso ela tenha sido satisfeita analisa-se a próxima restrição e assim sucessivamente.

Como o problema foi formulado utilizando-se cromossomos binários, como explicado anteriormente, trabalha-se com os operadores genéticos clássicos:

- *Crossover:* Sorteia-se dois indivíduos por *Roulette Wheel*, que serão os pais. O crossover uniforme é realizado com probabilidade P_c, o que significa que os pontos de crossover serão escolhidos segundo P_c. Os indivíduos resultantes(filhos) formarão uma população intermediária chamada de *pop_cross*.

- *Mutação:* Sorteia-se indivíduos por Roullete Wheel, dentre os indivíduos de *pop_cross*, e aplica-se mutação em seus genes com probabilidade P_m. Isto significa que o número de genes a ser mutado depende de P_m. Esses indivíduos modificados formarão a população intermediária chamada *pop_mut*.

- *Seleção:* Trabalhou-se com seleção elitista, mantendo os melhores indivíduos proporcionalmente ao tamanho da população. Os indivíduos que constituirão a nova população serão escolhidos dentre a antiga população, a população de crossover(*pop_cross*) e a população de mutação (*pop_mut*).

O Algoritmo Genético possui algumas variáveis que são decisivas para o bom funcionamento do mesmo. O tamanho da população, o número de gerações, o tamanho das populações intermediárias, o valor das penalidades etc, todos são fatores de grande importância e serão tratados posteriormente.

3.3.2 As Restrições

As restrições são parte muito importante da formulação do problema, são elas que limitam o espaço de busca e permitem que o algoritmo ande pelo caminho certo,

gerando soluções válidas para o problema. Nesta seção serão tratadas detalhadamente cada uma das restrições usadas no problema de otimização topológica de estruturas.

Restrição 1: Aceitabilidade da estrutura segundo seus nós básicos Esta restrição diz respeito à existência ou não dos nós básicos na estrutura. É uma restrição muito forte pois, sem estes nós, a estrutura não existe. Como já foi dito anteriormente, os nós básicos são aqueles que possuem algum tipo de força ou condição de contorno; por este motivo são essenciais na estrutura. Todas as soluções geradas devem possuir em sua composição os nós básicos. Caso eles não estejam presentes, o valor do *fitness* correspondente àquela solução é penalizado e as chances desse indivíduo continuar existindo nas próximas gerações é muito remota.

Restrição 2: Estabilidade da estrutura Nesta restrição verifica-se se a estrutura gerada pelo Algoritmo Genético é estável, ou seja, se não há nenhum mecanismo presente na estrutura. Para isto verifica-se a matriz de rigidez do indivíduo, a partir de sua conectividade. Se essa matriz for definida positiva, isto indica que a matriz é estável e que o sistema gerado pelo Método dos Elementos Finitos pode ser resolvido. Neste caso a restrição é satisfeita. Caso contrário, o sistema gerado pelo Método dos Elementos Finitos não poderá ser resolvido, o que indica que a estrutura gerada é um mecanismo e portanto entrará em colapso. Neste caso uma penalidade será aplicada ao *fitness* do indivíduo e as chances de sobrevivência serão pequenas.

Restrição 3: Deslocamento $(\delta_k^{max} - \delta_k(A, \xi) = 0, k = 1, 2, ..., n)$ Essa restrição diz respeito aos nós da estrutura, e impede que eles sofram um deslocamento maior que o permitido, de forma a não comprometer a estrutura final.

O deslocamento é obtido através da resolução do sistema linear:

$$[K]\{u\} = \{f\}$$

onde, $[K]$ é a matriz de rigidez excluindo-se os nós de sustentação, para os quais não ocorrem desvios, já que esses nós são fixos, $\{f\}$ é o vetor de forças aplicadas nos nós e $\{u\}$ é o deslocamento obtido.

Os valores de deslocamentos (δ_k) nos nós obtidos pela resolução do sistema linear são comparados, nó a nó, com o valor limite pré estabelecidos(δ_k^{max}). A penalidade será aplicada proporcionalmente ao desvio desses deslocamentos. Será calculado o valor absoluto de $\delta_k^{max} - \delta_k$ em cada um dos nós e depois somados e multiplicados por um fator definido anteriormente. O *fitness* do indivíduo será penalizado com o valor obtido.

Restrição 4: Resistência $(S_j - \sigma_j(A, \xi) = 0, j = 1, 2, ..., m)$ Nesta restrição serão verificadas as tensões resultantes sobre cada elemento da estrutura. A tensão resultante em cada elemento não poderá ser maior que o limite de resistência do material definido para o material usado pela estrutura.

A tensão (σ_j) em cada uma das barras é calculada utilizando-se os valores obtidos na resolução do sistema $[K]\{u\} = \{f\}$, e é dada por:

$$\sigma_j = \left(\frac{E}{l}\right) * (\cos(u_2 - u_1) + \sin(v_2 - v_1)) \tag{3.1}$$

onde, E é o Módulo de Elasticidade $[N/m^2]$ do material de cada barra, l é o comprimento de cada barra $[m]$ e u_i e v_i são os deslocamentos obtidos em cada nó nas direções x e y respectivamente $[m]$.

A verificação é feita para cada elemento da malha, e os valores relativos ao módulo de $S_j - \sigma_j$ para cada um desses elementos é somado e multiplicado por um fator determinado anteriormente. Esse resultado será usado para penalizar o *fitness* do indivíduo.

As restrições 3 e 4 não são restrições fortes, mas sim eliminatórias. A penalidade aplicada por elas não são muito grandes, a fim de não inviabilizar a manutenção do indivíduo na população, mas apenas distinguir indivíduos bons de indivíduos não muito bons.

No presente trabalho foram utilizados diferentes tipos de pesos para as penalidades. Num primeiro momento os pesos utilizados foram números muito grandes, da ordem de 10^9, porém, alguns problemas não obtiveram bom desempenho, pois este valor era muito maior ou muito menor do que a massa da estrutura. No primeiro caso, a penalidade não fazia diferença na escolha do indivíduo e no segundo caso a penalidade era tão grande que inviabilizava a escolha desse indivíduo, gerando falta de diversidade na população.

Num segundo momento os pesos foram relativos à massa total da estrutura, isto é, o programa calcula a massa da estrutura inicial e calcula os pesos proporcionalmente a essa massa total. Este método gerou bons resultados em todos os testes.

3.3.3 Algoritmo Final

Após a codificação do problema e a apresentação das restrições, esta seção apresenta uma visão mais ampla de todo o algoritmo. O algoritmo 2 apresenta o funcionamento geral do Algoritmo Genético implementado.

O algoritmo 3 mostra como a população inicial é gerada em um Algoritmo Genético binário. Alguns valores de P_i são testados no capítulo 5.

Algorithm 2 Algoritmo Genético implementado.

```
1: Inicializa população
2: Calcula fitness
3: while geração ≤ geração_máxima do
4:    Gera população de elite
5:    for tamanho = 1até tamanho de pop_cross do
6:       Seleciona pai1 da população
7:       Seleciona pai2 da população
8:       Realiza crossover
9:       Armazena filhos na população de crossover
10:   end for
11:   Calcula o fitness da população de crossover
12:   for tamanho = 1 até tamanho de pop_mut do
13:      Seleciona indivíduo da população de crossover
14:      Realiza mutação
15:      Armazena indivíduos mutados na população de mutação
16:   end for
17:   Calcula o fitness da população de mutação
18:   Gera nova população
19: end while
```

Algorithm 3 Algoritmo para gerar a população inicial

```
1: for i=1 até tamanho da população do
2:    for j=1 até tamanho do cromossomo do
3:       if número aleatório< P_i then
4:          pop(i,j)=0
5:       else
6:          pop(i,j)=1
7:       end if
8:    end for
9: end for
```

Capítulo 4

Implementação Computacional

4.1 Introdução

A implementação computacional dos modelos e a análise por elementos finitos bem como as rotinas para otimização estrutural usando Algoritmos Genéticos, foram feitas usando linguagem orientada a objetos em ambiente C++, tendo como objetivo o desenvolvimento de uma base ampla, tanto para os problemas de análise de engenharia como para as técnicas de otimização estrutural.

O Algoritmo Genético foi acoplado à base parcialmente implantada de análise de elementos finitos, denominada *MefLab++*.

O *MefLab++* é um conjunto de programas que está sendo desenvolvido por alunos e professores do Departamento de Mecânica Computacional da Faculdade de Engenharia Mecânica da Unicamp.

Neste capítulo faz-se inicialmente algumas considerações gerais sobre as fases do processo de desenvolvimento do *modelo orientado a objetos* para a resolução de um problema geral de otimização e na seqüência aplica-se o modelo ao caso do Algoritmo Genético Binário desenvolvido neste trabalho.

4.2 Programação Orientada a Objetos para Problemas de Otimização Evolutiva

A Programação Orientada a Objetos é baseada no conceito de se criar uma implementação computacional que melhor adapte o computador à resolução de um problema. Este conceito se opõe ao utilizado em programações estruturadas convencionais, que modelam o problema em algo mais apropriado à máquina. A Programação Orientada a Objetos tem como características principais um código portável e reutilizável, com encapsulamento do programa [29].

A estratégia de implementação da Programação Orientada a Objetos consiste em identificar os aspectos do seu problema relacionados a componentes físicos e conceituais, ou etapas a serem realizadas, que permitam a resolução do problema. Estes aspectos são organizados em pequenas partes, que podem vir a existir computacionalmente como entidades. Estas entidades são então representadas de forma única por objetos dentro do programa.

Todo o conhecimento de cada um dos objetos é armazenado em uma classe, que é definida por um conjunto de variáveis de tipos diferentes combinadas com um conjunto de funções relacionadas às variáveis. As classes também podem ser compostas por outros tipos de objetos.

Estas classes podem estar relacionadas com outras classes através da utilização de heranças. Este relacionamento é caracterizado pela identificação de similaridades e diferenças entre objetos. Pode-se também estabelecer relações de uso e de posse [21].

Assim sendo, tem-se que as semelhanças entre objetos são agrupadas em classes de base ou principais, ao passo que as diferenças são definidas em classes derivadas. Estas classes derivadas podem ser descritas como classes especializadas que herdam da classe de base todas as especificações, como métodos e variáveis, e ainda podem apresentar características próprias.

Uma vez definidas as classes, a Programação Orientada a Objetos compõe o problema em um conjunto de objetos, que cooperam entre si. Este relacionamento entre os objetos se dá através dos métodos da classe da qual eles foram instanciados. Assim sendo, os objetos são criados e mandam mensagens uns para outros, recebendo e distribuindo tarefas, e após cada um cumprir o seu propósito, o problema estará resolvido.

Em resumo, pode-se destacar três características principais da Programação Orientada a Objetos, que são:

- ter nos objetos, e não nos algoritmos, a base da criação de programas;
- cada objeto é a instância de uma classe, com a sua própria identidade, que serve para diferenciá-lo de outros objetos que possuam os mesmos conjuntos de operações;
- o relacionamento entre os objetos se dá através dos métodos das classes das quais eles foram instanciados.

4.3 Obtendo um Modelo de Objeto

Um bom modelo de objeto é conseguido através de um processo de desenvolvimento que possui três etapas, como segue [12]:

1. Análise Orientada a Objetos: é a fase onde um especialista descreve o problema em termos das necessidades e tarefas a serem resolvidas pelo programador.

2. Projeto Orientado a Objetos: é a fase de definição das classes e de suas variáveis e funções que irão fazer a vinculação e a comunicação entre os objetos.

3. Programação Orientada a Objetos: é a fase de implementação, onde se tem a codificação dos métodos e funções, os testes e a integração do programa como um todo.

Em seguida tem-se a aplicação das duas primeiras etapas descritas acima, no desenvolvimento do modelo de objeto para a otimização estrutural, usando Algoritmos Genéticos.

Para auxiliar no desenvolvimento do programa *MefLab++*, utilizou-se uma base integrada de desenvolvimento chamada *Rational Rose*, que é um programa de engenharia de *software*, que permite ao programador manter uma apresentação mais clara do programa em desenvolvimento.

No programa Rational Rose são definidas todas as classes do *MefLab++*, seus atributos bem como os métodos de cada uma das classes. Após gerada a estrutura de base, a implementação é feita no Visual C++ diretamente.

Várias notações são usadas para auxiliar o programador a criar as classes e especificar as relações e atributos das mesmas. Neste trabalho usou-se a notação proposta por *"Booch"*, implementada no programa Rational Rose, que é a ferramenta básica usada para a criação e manutenção das classes C++ desenvolvidas nesse trabalho.

As classes e suas relações de herança, conforme essa notação, são indicadas na Figura 4.1.

4.3.1 Análise Orientada a Objetos

A análise orientada a objetos do módulo de otimização estrutural tem como objetivo especificar o problema a ser resolvido, posicioná-lo em relação ao programa *MefLab++* já implementado e identificar as etapas a serem realizadas.

Os problemas de otimização estrutural podem ser resolvidos usando-se basicamente duas famílias de métodos: os métodos heurísticos e os métodos baseados em programação matemática.

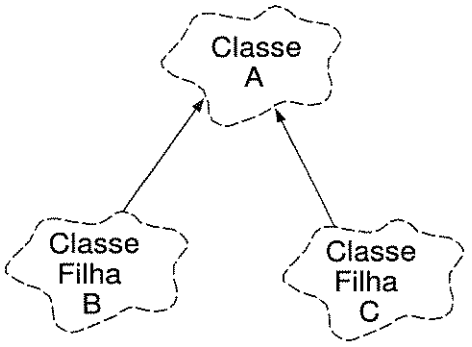


Figura 4.1: Diagrama de classe genérica na notação *Booch*.

Um primeira análise permitiu identificar que o problema poderia ser decomposto em duas classes: *"Domain"* e *"Tasks"*, conforme mostrado na Figura 4.2:

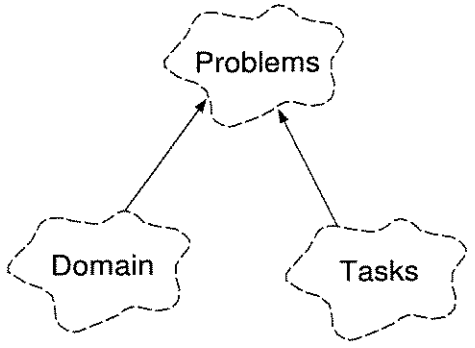


Figura 4.2: Diagrama inicial do *MefLab++*.

A classe *"Domain"* é responsável pela representação do modelo mecânico a ser utilizado. A classe *"Tasks"* é responsável por executar as tarefas que o usuário especificar. Assim, a implementação do Algoritmo Genético, estará concentrada na classe *"Tasks"*.

Ainda de forma geral, a classe *"Tasks"* foi desenvolvida de forma a contemplar diversas possibilidades de análise e também variadas possibilidades de otimização. O diagrama de classes que apresenta a estrutura adotada é mostrado na Figura 4.3.

O objetivo deste trabalho concentrou-se no desenvolvimento do módulo *"Genetic Algoritm"*, onde foram necessários realizar as tarefas listadas abaixo.

Inicialização dos parâmetros e dados do modelo. Alguns parâmetros são obtidos a partir de um arquivo de dados, desta forma eles serão alterados a cada exemplo estudado.

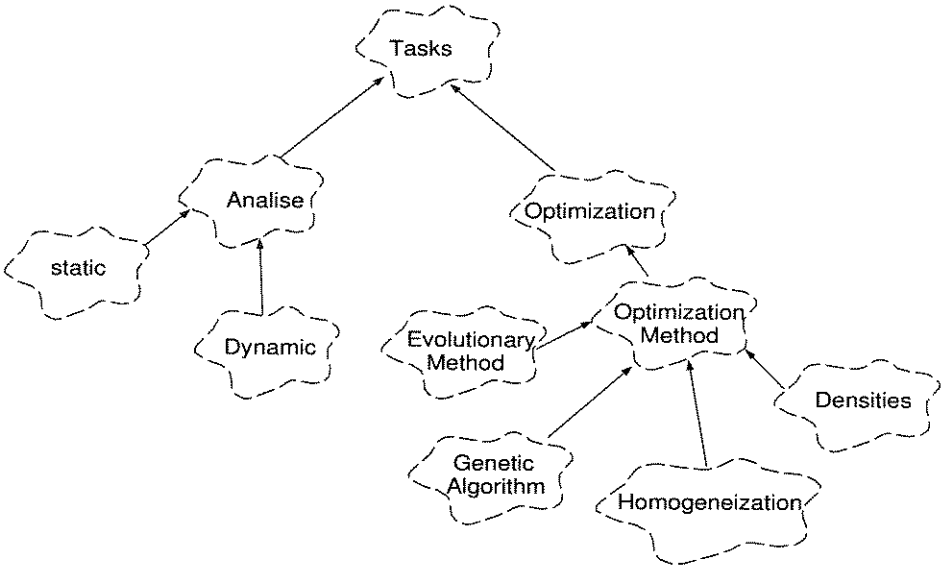


Figura 4.3: Diagrama de classes utilizado pelo *MefLab++*.

1. Parâmetros relacionados ao modelo:

   - *max_bar* = número máximo de barras na treliça.
   - *Area* = área da seção transversal das barras.
   - *E* = módulo de Elasticidade
   - *max_disp* = deslocamento máximo admitido para qualquer nó da estrutura.
   - *max_stress* = tensão máxima admitida.
   - *conectivity* = matriz de conectividade dos elementos.
   - *NodeList*= vetor com os nós básicos a serem mantidos no modelo.
   - *GeometricNodes* = coordenadas nodais.
   - *Force* = forças aplicadas no modelo.
   - *Displacement* = deslocamento imposto no modelo.

2. Parâmetros relacionados ao algoritmo:

   - *size_pop* = tamanho da população.
   - *num_ger* = número máximo de gerações.
   - *prob_cross* = probabilidade de crossover.

- *prob_mut* = probabilidade de mutação.

Os parâmetros referentes ao modelo mudam a cada exemplo que será testado, para que os dados se ajustem e descrevam com exatidão o modelo pretendido. Já os dados do algoritmo determinam as características do Algoritmo Genético, independentemente do modelo que está sendo rodado. No Apêndice B encontra-se um exemplo do arquivo de entrada de dados utilizado no *MefLab++*.

Cada classe possui um método especializado na leitura dos dados do arquivo. Este arquivo permite definir o domínio e a tarefa que se deseja executar sobre o respectivo domínio, completamente.

Geração da população inicial de cromossomos. O cromossomo inicial na população binária, consiste de valores binários 0 ou 1 gerados aleatoriamente. O algoritmo 3 apresentado na seção 3.3.3 descreve a geração da população inicial no Algoritmo Genético.

Cálculo do valor de fitness. Nessa etapa avalia-se o fitness da população, seja ela a população inicial ou a sub-população encontrada após aplicar-se os operadores genéticos. Deve-se lembrar mais uma vez que o fitness vai depender, neste caso, das penalidades aplicadas às restrições não satisfeitas e à massa da estrutura. Para a aplicação do Roulette Wheel fez-se uma inversão e uma normalização no vetor de fitness, obtendo desta forma uma melhor proporção na representação de cada indivíduo. O algoritmo 1, utilizado para a aplicação das penalizações, foi apresentado na seção 3.3.1.

Evolução das gerações. Nessa fase aplica-se os operadores de mutação, crossover e seleção como foi explicado anteriormente, para realizar-se a evolução das gerações.

Analisa-se os melhores indivíduos ordenados pelo fitness e constrói-se com esses cromossomos uma sub-população denominada *pop_elite*.

Seleciona-se em seguida os indivíduos para mutação e crossover através de Roulette Wheel e processo aleatório. Aplica-se a mutação e crossover nestes indivíduos.

Após realizar-se a mutação e o crossover, tem-se duas novas sub-populações, *pop_mut* e *pop_cross*, originárias da mutação e crossover, respectivamente.

A nova população irá consistir então dos indivíduos selecionados de *pop_elite*, *pop_mut* e *pop_cross*.

Atualiza-se o contador de gerações *num_ger*.

Escolha do melhor cromossomo. Através do valor da função de fitness encontra-se o melhor cromossomo da nova população.

Próxima Geração. Caso não se tenha atingido o número máximo de gerações, *num_ger*, repete-se o processo. Caso contrário, finaliza-se o algoritmo e tem-se, neste caso, o conjunto de soluções ótimas obtido para esse número de gerações.

4.3.2 Projeto Orientado a Objetos

O projeto orientado a objetos tem o objetivo de identificar prováveis objetos e especificar as classes das quais serão instanciados estes objetos, sendo que, o exercício de caracterização de objetos é feito através da abstração das classes.

A abstração pode ser resumida como o fato de separar mentalmente um ou mais elementos de uma totalidade complexa como, por exemplo, um problema de otimização evolucionária a ser desenvolvido, de modo a concentrar-se na generalização de seu comportamento, ignorando seus detalhes.

Portanto, a função da abstração é reconhecer cada um dos objetos e seus métodos através da definição de conceitos e características essenciais, que irão diferenciá-los dos demais. Desta forma, a abstração permite visualizar de forma global a interface do objeto, separando o seu comportamento da sua implementação.

A interface do objeto pode ser definida como a sua forma de interação com os outros objetos do projeto, sendo esta, feita através dos métodos, que têm a função de definir como o objeto vai agir e reagir.

Conforme mencionado anteriormente o modelo de objeto do *MefLab++* está definido basicamente como um problema a ser resolvido, que é composto de um domínio e uma tarefa a ser cumprida.

O domínio, neste caso, caracteriza-se por uma equação diferencial válida em uma certa região geométrica e sujeita a condições de contorno. Já a tarefa é algo a ser realizado com respeito ao domínio, como por exemplo uma análise estrutural dinâmica ou estática.

O programa *MefLab++* é constituído basicamente de um objeto representando o problema a ser resolvido. Este objeto contém mais dois objetos, o objeto instanciado da classe *Domain* referindo-se ao domínio do problema e o objeto instanciado da classe *Tasks* que representa a tarefa a ser realizada sobre o domínio.

Desta forma considerou-se a otimização estrutural como uma tarefa do problema, resultando na abstração de um objeto instanciado da classe *Optimization*. Considerando a diversidade dos métodos de otimização, instanciou-se um objeto *theOptimizationMethod*, que caracteriza o método de resolução adotado.

Neste trabalho concentrou-se na implementação das classes ligadas ao objeto *theGeneticAlgorithm*. Da análise dos Algoritmos Genéticos é possível observar que diferentes tipos de codificações genéticas conduzem a algoritmos diversos.

Estas diferentes codificações determinaram a necessidade de objetos que carac-

terizem cada algoritmo a ser considerado, determinando-se, inicialmente, três tipos diferentes de objetos: *theIntegerGA*, *theBinaryGA* e *theRealGA*.

A classe *IntegerGA*, ainda não implementada, conterá as informações necessárias para a execução de um Algoritmo Genético inteiro. Nela estarão os operadores genéticos necessários para este tipo de algoritmo. A classe *RealGA*, também ainda não implementada, conterá as informações necessárias para a execução de um Algoritmo Genético com código real.

A classe *BinaryGA* contém as operações básicas de um Algoritmo Genético com código binário. Esta classe possui os operadores de mutação, seleção e crossover específicos para códigos binários.

As três classes acima, *IntegerGA*, *BinaryGA* e *RealGA*, são classes filhas da classe *GeneticAlgorithm*, que contém todas as operações necessárias para a realização de um algoritmo genético. Esta é uma classe virtual, que significa que gerencia o Algoritmo Genético, direcionando-o para o tipo de algoritmo escolhido: inteiro, real ou binário. Na classe *GeneticAlgorithm* estão os métodos comuns aos três tipos de codificação.

O diagrama mostrado na Figura 4.4 esclarece as relações estabelecidas com a classe *GeneticAlgorithm*.

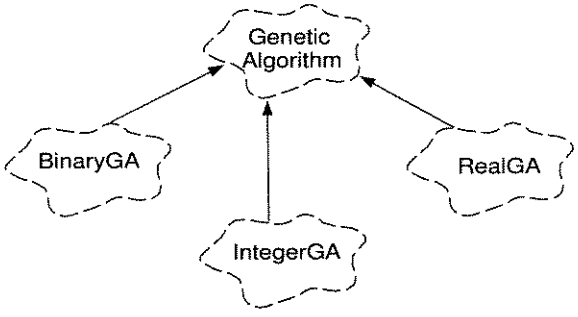


Figura 4.4: Diagrama representando as classes filhas da classe *GeneticAlgorithm*

A classe *GeneticAlgoritm* possui algumas operações virtuais(*crossover*, *mutation*, *selection* e *initial_ pop*). Estas operações não estão implementadas na classe *GeneticAlgoritm*, mas sim em seus filhos. Estas operações são redirecionadas ao tipo de algoritmo escolhido para ser executado. Além dessas operações virtuais, a classe possui uma operação chamada *sort* que ordena o fitness dos indivíduos e retorna um vetor de índices com esta ordenação. Este vetor é usado pelo programa para criar a população de elite.

A classe *BinaryGA*, além de possuir a implementação das operações de crossover, mutação, seleção e criação da população inicial, para cromossomos binários, possui a operação *give_ pop_elite*, que gera a população de elite e *give_ new_ pop*, que gera

a nova população.

A Figura 4.5 mostra a janela aberta pelo *Rational Rose* onde estão as operações da classe *GeneticAlgoritm*.

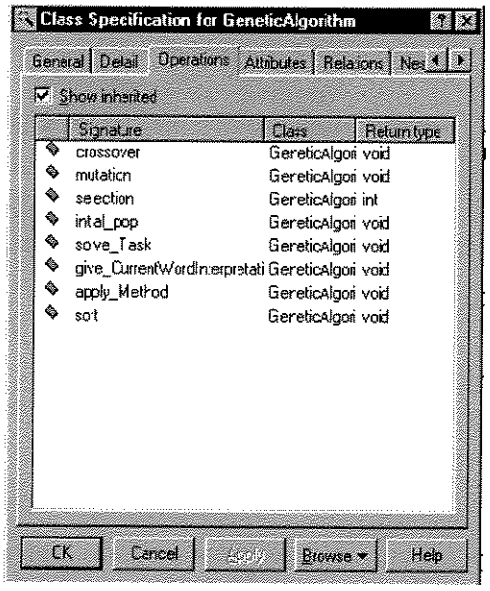


Figura 4.5: Operações da classe *GeneticAlgoritm*.

Além do bloco de classes que gerenciam o Algoritmo Genético, foi necessária a criação de um bloco de classes para gerenciar o cálculo do fitness. O fitness é calculado em duas partes: valor da função objetivo e valor da penalidade aplicada devido às restrições. Desta forma foram criadas mais três classes: *Fitness*, *ObjetiveFunction* e *Constraints*. Veja na Figura 4.6 como ficaram as relações entre essas classes.

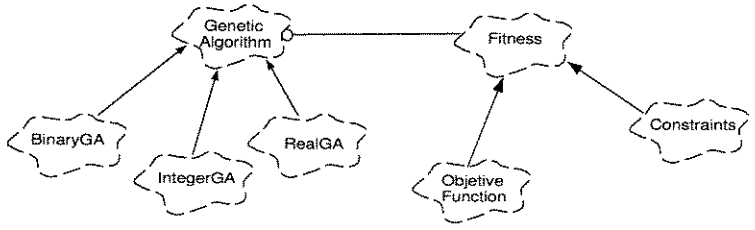


Figura 4.6: Diagrama representando as classes filhas da classe *Fitness*

A classe *Fitness*, que calcula o fitness total de cada indivíduo da população, possui uma relação de "uso" com a classe *GeneticAlgorithm*, o que significa que a classe *GeneticAlgorithm* usa os resultados obtidos pela classe *Fitness* para a real-

ização de suas operações, e possui as classes *ObjetiveFunction* e *Constraints* como classes filhas.

A classe *Fitness* possui apenas uma operação virtual: *calc_fitness*. Esta operação gerencia todo o cálculo do fitness, que é feito através das classes *ObjectiveFunction* e *Constraints*. A Figura 4.7 mostra a janela aberta pelo *Rational Rose* onde estão as operações da classe *Fitness*.

Observa-se que pode-se ter vários tipos de função objetivo e de restrições. Desta forma, a classe *ObjetiveFunction* possui duas classes filhas: *TotalMass*, que tem como função objetivo a massa total da estrutura, e *Compliance*, que tem como função objetivo a compliância da estrutura.

A classe *ObjetiveFunction* possui uma operação virtual chamada *calc_objective_function*, que redireciona a operação para a função objetivo escolhida.

A classe *Constraints* possui quatro classes filhas: *DisplacementLimit*, *BasicNodes*, *Singularity* e *StressLimit*. Cada uma dessas classes verifica uma restrição do problema. O problema pode ou não ter essas quatro restrições. A classe *DisplacementLimit* tem como restrição o deslocamento nos nós da treliça, a classe *BasicNodes* verifica a presença dos nós básicos na treliça, a classe *Singularity* verifica se a matriz de rigidez da estrutura é definida positiva ou não e a classe *StressLimit* tem como restrição a tensão nas barras da treliça.

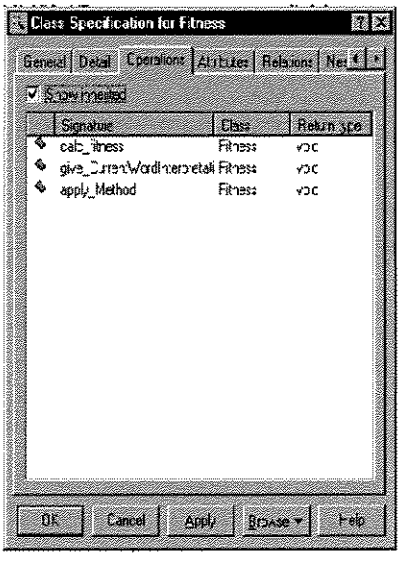


Figura 4.7: Janela do Rational Rose

A classe *Constraints* possui uma operação virtual chamada *calc_penalty*. Esta operação é direcionada para cada uma das restrições do problema. Portanto, existe

uma operação chamada *calc_penalty* em cada uma das restrições: *BasicNodes*, *Singularity*, *DisplacementLimit* e *StressLimit*. Dependendo da classe a que se refere, a operação retorna o valor da penalidade em questão.

A Figura 4.8 mostra a estrutura completa do projeto orientado a objetos que representa o módulo de Algoritmo Genético. As classes não comentadas neste trabalho já existiam no sistema e foram ligadas às novas classes para possibilitar o uso de algumas de suas operações.

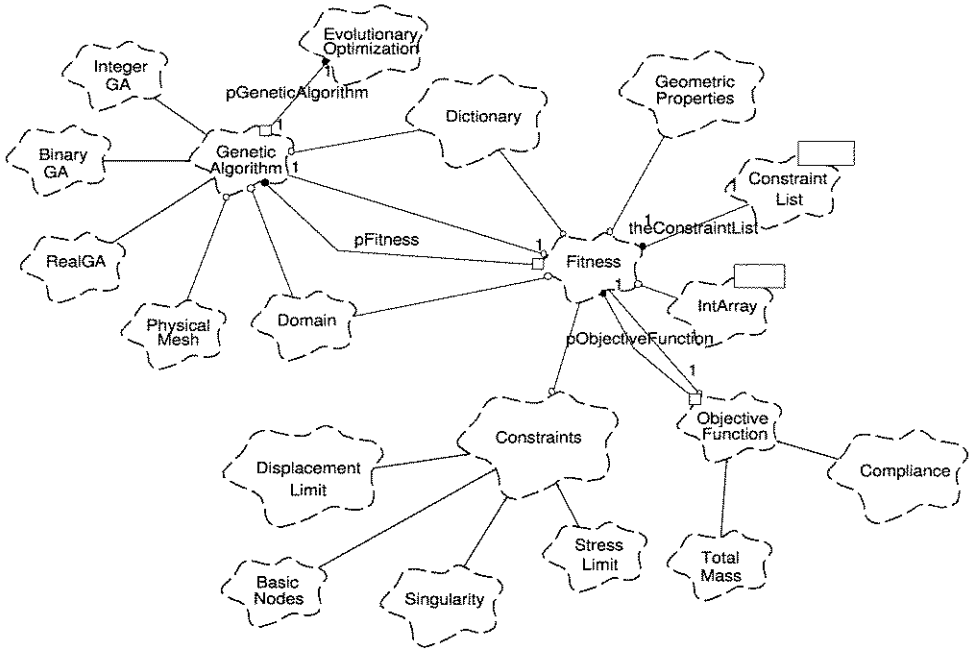


Figura 4.8: Diagrama feito no Rational Rose

Para tornar possível o cálculo das penalidades, incluiu-se algumas novas operações nas classes já existentes no sistema. Para penalizar o fitness em relação à tensão criou-se na classe *Domain* uma operação chamada *give_stressVector*, que fornece o vetor das tensões calculadas. Para penalizar o fitness em relação ao deslocamento criou-se na classe *Domain* uma operação chamada *give_dispVector*, que fornece o vetor dos deslocamentos calculados. Já para a verificação da singularidade da matriz de rigidez, criou-se uma operação chamada *checkSingularity*, também no *Domain*, que verifica se a matriz é singular através de seus autovalores. Para checar a existência dos nós básicos na estrutura, criou-se no *Domain* uma operação chamada *checkGeoNode*, que verifica se os nós básicos estão entre os nós geométricos da treliça. Outra alteração importante para o funcionamento do algoritmo foi a criação de uma função que modificaria a lista de nós físicos do problema. Como o programa gera

várias possíveis soluções para o problema e é preciso avaliar essas soluções, para que a avaliação seja possível foi preciso atualizar o sistema com as configurações de cada solução, isto é, acrescentar e retirar elementos da estrutura para depois calcular a função objetivo e verificar as restrições. Para isso foram criadas as funções *update_PhysicalMesh* no *Domain* (que atualiza a malha de elementos) e *Reset_list* na classe *PhyElement* (que recupera a lista de elementos original).

Capítulo 5

Resultados Numéricos e Análise de Desempenho

Neste capítulo apresentam-se os resultados obtidos ao longo do trabalho. Serão apresentados os estudos realizados para a calibração de parâmetros, algumas estatísticas e os exemplos práticos resolvidos. Inicialmente serão apresentados os resultados obtidos para estruturas reticuladas e posteriormente alguns experimentos numéricos são mostrados usando-se o modelo de estado plano de tensões com pequenos deslocamentos.

Os exemplos escolhidos são clássicos da literatura, a fim de validar o algoritmo. Todos os exemplos têm como critério de otimização a minimização da massa total da estrutura, obedecendo às restrições que dizem respeito aos deslocamentos máximos nos nós e às tensões máximas nas barras.

As malhas iniciais utilizadas nos exemplos foram geradas pelo software ANSYS e a visualização dos resultados foi obtida através de um programa em Matlab, visto que o desenvolvimento de um sistema de pré e pós processamento ainda não foram concluídos pelo grupo de trabalho dentro do programa *MefLab++*.

Em todos os exemplos testados as hipóteses de que o material é linear, isotrópico e homogêneo são assumidas.

5.1 Exemplos estudados

Com o objetivo de avaliar o comportamento do algoritmo implementado, bem como validar a implementação computacional efetuada, foram estudados os seguintes casos.

Inicialmente usou-se um exemplo simples, estudado por diversos autores ([8] [35]), de uma estrutura treliçada engastada livre sujeita a um carregamento de flexão. O exemplo é simples, todavia representativo, e devido ao seu baixo custo computa-

cional, permitiu realizar alguns experimentos numéricos visando a calibração dos parâmetros do algoritmo, bem como um estudo estatístico sobre o comportamento do mesmo.

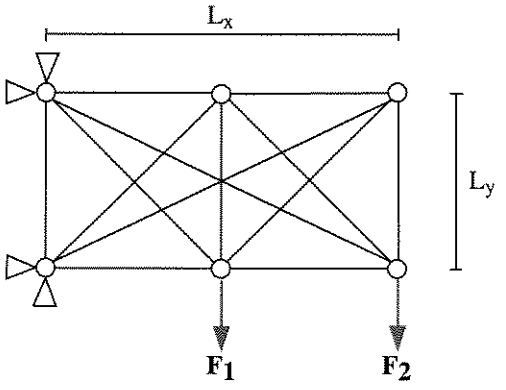


Figura 5.1: Treliça utilizada como exemplo 1.

Na figura 5.1, mostra-se a malha inicial do exemplo 1, treliça engastada livre, onde todas as barras que constituem o espaço de trabalho foram incluídas.

Para este exemplo foram estudados dois casos, cujos dados são mostrados na Tabela 5.1.

Tabela 5.1: Casos estudados no Exemplo 1.

Casos	Caso a	Caso b
$L_x(m)$	7.2	2.0
$L_y(m)$	3.6	1.0
Área(m^2)	0.0025	0.0025
$F_1(N)$	-100000	-5000
$F_2(N)$	-100000	-5000
$E(N/m^2)$	2.1×10^{11}	2.1×10^{11}
n. de nós	6	6
n. inicial de elementos	13	13
deslocamento máximo (m)	0.01	0.01
tensão máxima (Pa)	100×10^6	100×10^6

O segundo exemplo estudado, é o de uma treliça bi-apoiada, sujeita a um carregamento de flexão. Este exemplo possui simetria diedral, e sua configuração inicial é mostrada na figura 5.2.

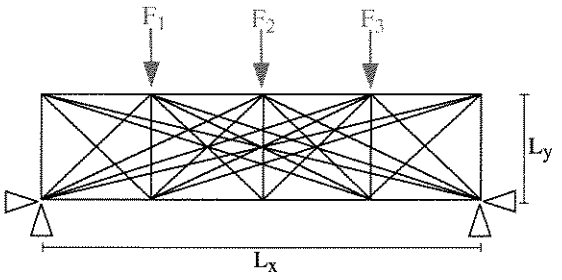


Figura 5.2: Treliça utilizada como exemplo 2.

Neste exemplo foram estudados dois casos, o primeiro com a estrutura total e o segundo aproveitando a simetria, cujos dados são mostrados na Tabela 5.2.

Tabela 5.2: Casos estudados no Exemplo 2.

Casos	Caso a	Caso b
$L_x(m)$	10.16	5.08
$L_y(m)$	2.54	2.54
Área(m^2)	2.5×10^{-3}	2.5×10^{-3}
$F_1(N)$	-200000	-200000
$F_2(N)$	-200000	-100000
$F_3(N)$	-200000	0
$E(N/m^2)$	6.5×10^{10}	6.5×10^{10}
n. de nós	10	6
n. inicial de elementos	45	13
deslocamento máximo (m)	0.05	0.05
tensão máxima (Pa)	162×10^6	162×10^6

O terceiro exemplo apresenta um grau de complexidade mais elevado, com uma topologia mais complexa, mas também possui simetria diedral, fato que pode ser explorado pelo usuário. Na figura 5.3 apresenta-se o domínio total de trabalho para a treliça do exemplo 3.

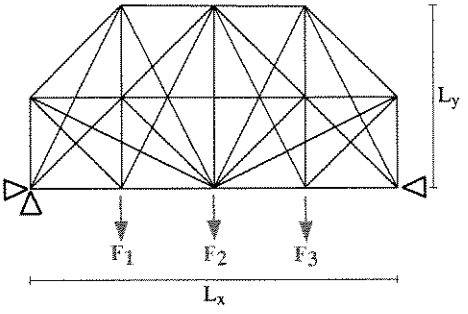


Figura 5.3: Treliça utilizada como exemplo 3.

Para este exemplo também foram estudados dois casos, o primeiro com a estrutura total e o segundo aproveitando a simetria, cujos dados são mostrados na Tabela 5.3.

Tabela 5.3: Casos estudados no Exemplo 3.

Casos	Caso a	Caso b
$L_x(m)$	12.192	6.096
$L_y(m)$	6.096	6.096
Área(m^2)	0.05	0.05
$F_1(N)$	-88964	-88964
$F_2(N)$	-88964	-44482
$F_3(N)$	-88964	0
$E(N/m^2)$	6.5×10^{10}	6.5×10^{10}
n. de nós	13	8
n. inicial de elementos	39	20
deslocamento máximo (m)	0.05	0.05
tensão máxima (Pa)	162×10^6	162×10^6

O quarto exemplo refere-se a um modelo de meio contínuo, particularmente o problema de equilíbrio de um meio elástico plano, onde são adotadas as hipóteses de estado plano de tensões.

Na figura 5.4 é apresentado o espaço inicial de trabalho, com suas respectivas condições de contorno.

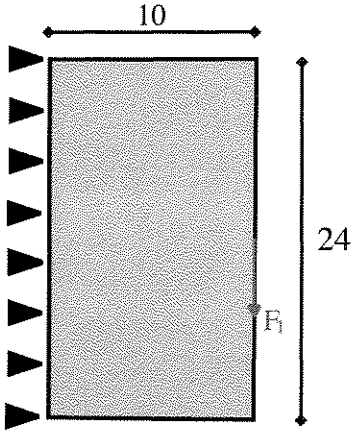


Figura 5.4: Estrutura de duas barras.

As propriedades do exemplo 4 são mostradas na Tabela 5.4.

Tabela 5.4: Propriedades do exemplo 4.

$L_x(m)$	10
$L_y(m)$	24
$F_1(N)$	-100000
$E(N/m^2)$	100^9
n. de nós	275
n. inicial de elementos	240
deslocamento máximo (m)	0.5
tensão máxima (Pa)	200^6

O quinto e último exemplo também refere-se a um modelo de meio contínuo, onde são adotadas as hipóteses de estado plano de tensões.

Na figura 5.5 é apresentado o espaço inicial de trabalho, com suas respectivas condições de contorno. Este exemplo aproveita a simetria do problema, utilizando-se apenas de metade da malha.

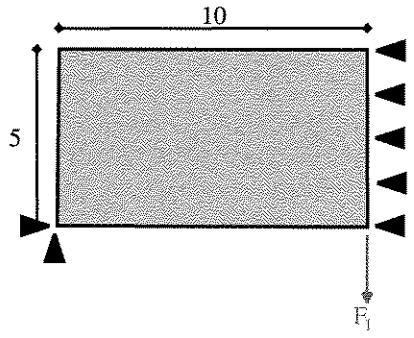


Figura 5.5: Estrutura de Michell.

As propriedades do exemplo 5 são mostradas na Tabela 5.5.

Tabela 5.5: Propriedades do exemplo 5.

$L_x(m)$	10
$L_y(m)$	5
$F_1(N)$	-625
$E(N/m^2)$	100^9
n. de nós	121
n. inicial de elementos	100
deslocamento máximo (m)	0.001
tensão máxima (Pa)	162^6

Para todos os exemplos mostrou-se soluções de massa (volume) mínimas sujeitas às condições de deslocamentos e tensões admissíveis.

5.2 Repetibilidade e Aspectos Estatísticos do Algoritmo Genético.

Antes de apresentar os resultados de cada exemplo, mostra-se um estudo estatístico preliminar, que permite caracterizar e situar melhor as soluções numéricas obtidas.

Considerando-se as características estocásticas do algoritmo implementado, uma das primeiras questões a ser colocada, refere-se à repetibilidade do processo.

A repetibilidade está diretamente ligada ao problema do critério de parada, que neste caso é o número de gerações.

Assim, pesquisou-se sobre a influência do número de gerações no resultado final obtido. Levantou-se uma estatística sobre o número de vezes em que o programa converge para uma mesma estrutura final após ser executado 20 vezes. O problema escolhido para ser testado foi o descrito no exemplo 1 mostrado na Figura 5.1, com as características mostradas no *caso b* da Tabela 5.1. Os parâmetros usados no algoritmo: P_i probabilidade de não ter elementos na estrutura inicial, P_c probabilidade de *crossover* e P_m probabilidade de mutação, são descritos na Tabela 5.6.

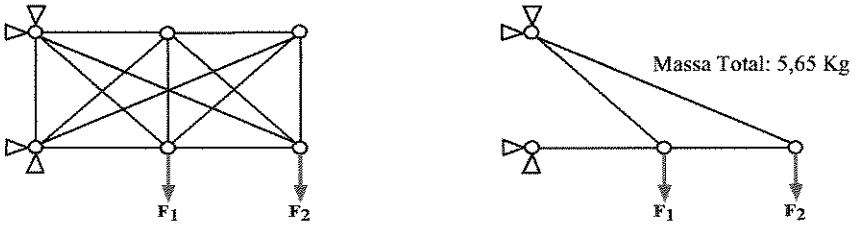
Tabela 5.6: Parâmetros - Exemplo 1 Caso b.

tamanho da população	20
P_i	0.4
P_c	0.6
P_m	0.1

Adotou-se inicialmente um esquema de *crossover* de dois pontos.

Como esperado, conforme aumentou-se o número de gerações obteve-se um maior número de melhores soluções. Nas figuras 5.7 e 5.8 as soluções são classificadas segundo a massa total da estrutura final obtida pelo algoritmo para diferentes números de gerações.

Na figura 5.6 mostra-se o espaço inicial de trabalho e a solução ótima obtida para este caso. Os aspectos físicos deste exemplo serão discutidos nas próximas seções.



Estrutura Inicial Estrutura Final

Figura 5.6: Figura comparativa - Exemplo 1.

Observando-se os gráficos das figuras 5.7 e 5.8, pode-se perceber que no gráfico da figura 5.7 o algoritmo chegou em três tipos de resultados diferentes, apenas 35% dos testes chegaram no resultado de menor massa e 65% das vezes em resultados piores. Aumentando-se para 400 gerações o quadro é bem diferente. Percebe-se no gráfico da figura 5.8 apenas dois tipos de resultados e 75% das vezes o algoritmo chegou na estrutura de menor massa.

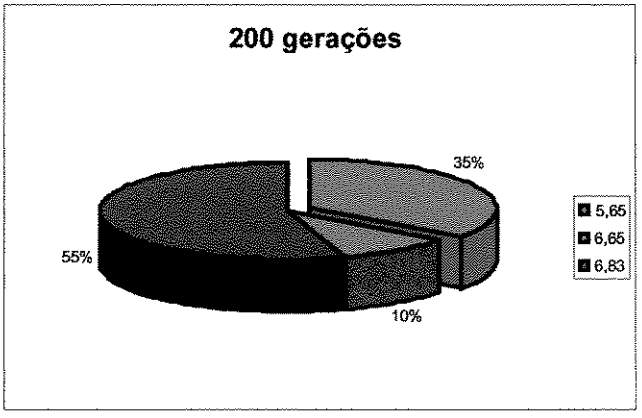


Figura 5.7: Gráfico Estatístico - *Crossover* de 2 pontos.

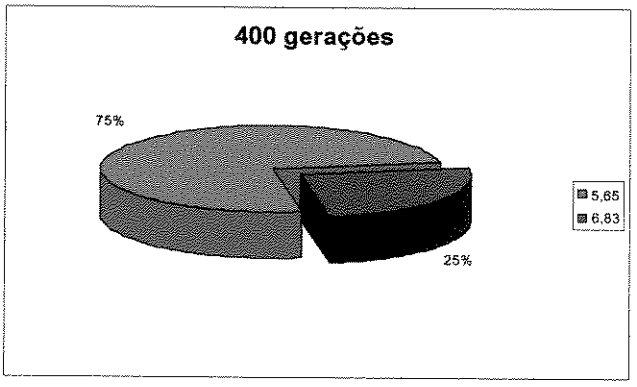


Figura 5.8: Gráfico Estatístico - *Crossover* de 2 pontos.

Este estudo foi repetido para outros exemplos, bem como para diferentes valores de P_i, P_c e P_m, e a tendência de se encontrar um maior número de soluções ótimas à medida que se aumenta o número de gerações foi confirmada. Cabe salientar no entanto, que resta sempre a possibilidade de se convergir para soluções intermediárias, que do ponto de vista da engenharia, podem ser aproveitadas. Este comportamento geral é freqüentemente reportado na literatura.

Realizou-se também um estudo sobre a influência do tipo de operador de *crossover* sobre a convergência do algoritmo. Escolheu-se o *crossover* uniforme e de dois pontos para serem testados [23]. Executou-se o mesmo número de vezes os dois tipos de *crossover* e observou-se quantas vezes cada um deles chegou no melhor resultado.

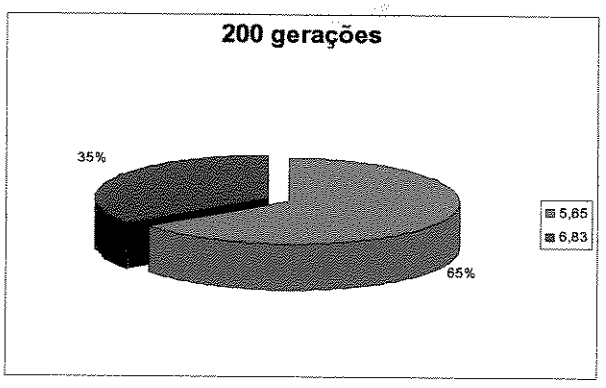


Figura 5.9: Gráfico Estatístico - *Crossover* Uniforme.

No gráfico da figura 5.9, obteve-se 65% de resultados com a menor massa, em contrapartida com os 35% da figura 5.7 obtidos pelo *crossover* de dois pontos. Logo,

conclui-se que o *crossover* uniforme possui um melhor desempenho para o caso analisado.

A partir dos estudos introdutórios realizados, nota-se que os parâmetros do algoritmo influenciam significativamente no seu desempenho. Por outro lado, o número de parâmetros envolvidos é grande, e a sua influência pode variar para cada caso estudado. No entanto, com base nos casos estudados, bem como na literatura consultada é possível afirmar que à medida em que se amplia o número de gerações, o Algoritmo Genético tende a convergir para uma solução ótima única. Assim, uma recomendação prática, quando se está abordando um novo problema, é procurar variar o número de gerações, acompanhando o comportamento das soluções encontradas.

Com base no mesmo tipo de argumentação, pode-se afirmar que o *crossover* uniforme mostrou-se superior ao *crossover* de dois pontos. Este resultado é específico para este tipo de problema.

5.3 Calibração de Parâmetros

Além do número de gerações e o tipo de *crossover*, fez-se um estudo estatístico visando a calibração dos parâmetros P_m Probabilidade de mutação, P_c Probabilidade de *crossover* e P_i Probabilidade inicial de zeros e uns. Este estudo teve como base uma população de tamanho 40, isto é, a população é formada por 40 treliças e adotou-se 2000 gerações.

O programa foi executado para os três problemas de estrutura reticuladas e seus variantes. Para cada exemplo foram utilizadas as combinações apresentadas na Tabela 5.7.

Tabela 5.7: Variação utilizada para os parâmetros.

Parâmetro	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
Pi	0,4	0,4	0,4	0,4	0,4	0,4	0,5	0,5	0,5
Pc	0,6	0,6	0,7	0,7	0,8	0,8	0,6	0,6	0,7
Pm	0,1	0,01	0,1	0,01	0,1	0,01	0,1	0,01	0,1
Parâmetro	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}
Pi	0,5	0,5	0,5	0,6	0,6	0,6	0,6	0,6	0,6
Pc	0,7	0,8	0,8	0,6	0,6	0,7	0,7	0,8	0,8
Pm	0,01	0,1	0,01	0,1	0,01	0,1	0,01	0,1	0,01

Os resultados estão nas Tabelas 5.8, 5.9, 5.10, 5.11, 5.12.

Na Tabela 5.8, tem-se os resultados obtidos para o exemplo 1 Caso *a*. Este exemplo convergiu para o resultado de menor massa em todos os conjuntos de parâmetros testados. O que variou foi apenas o número de iterações efetuadas para se chegar

no melhor resultado. Alguns conjuntos de parâmetros obtiveram o resultado em um número reduzido de iterações, como por exemplo o caso de $P_i = 0,4$, $P_c = 0,7$ e $P_m = 0,01$, onde o ótimo foi encontrado na quarta iteração.

Tabela 5.8: Calibração de parâmetros: Exemplo 1 Caso a.

Pi	Pc	Pm	Iteração Ótimo	Massa Final(Kg)	Penalização
0,4	0,6	0,1	27	508,43	0
0,4	0,6	0,01	225	508,43	0
0,4	0,7	0,1	194	508,43	0
0,4	0,7	0,01	4	508,43	0
0,4	0,8	0,1	181	508,43	0
0,4	0,8	0,01	897	508,43	0
0,5	0,6	0,1	424	508,43	0
0,5	0,6	0,01	216	508,43	0
0,5	0,7	0,1	130	508,43	0
0,5	0,7	0,01	610	508,43	0
0,5	0,8	0,1	519	508,43	0
0,5	0,8	0,01	859	508,43	0
0,6	0,6	0,1	146	508,43	0
0,6	0,6	0,01	678	508,43	0
0,6	0,7	0,1	271	508,43	0
0,6	0,7	0,01	260	508,43	0
0,6	0,8	0,1	271	508,43	0
0,6	0,8	0,01	412	508,43	0

Na Tabela 5.9, tem-se os resultados obtidos para o exemplo 1 Caso *b*. Neste caso, o exemplo também convergiu para o resultado de menor massa em todos os conjuntos de parâmetros testados, variando-se apenas a iteração em que se chegou no melhor resultado. Observa-se que no caso *a* o algoritmo executou uma média de 351 iterações para obter o ótimo, enquanto que no caso *b* essa média foi de 252 iterações. Isto mostra que as propriedades do exemplo são bastante influentes no resultado, mesmo que o layout inicial seja igual.

Tabela 5.9: Calibração de parâmetros: Exemplo 1 Caso b.

Pi	Pc	Pm	Iteração Ótimo	Massa Final(Kg)	Penalização
0,4	0,6	0,1	280	5,65	0
0,4	0,6	0,01	623	5,65	0
0,4	0,7	0,1	183	5,65	0
0,4	0,7	0,01	58	5,65	0
0,4	0,8	0,1	42	5,65	0
0,4	0,8	0,01	887	5,65	0
0,5	0,6	0,1	121	5,65	0
0,5	0,6	0,01	287	5,65	0
0,5	0,7	0,1	225	5,65	0
0,5	0,7	0,01	89	5,65	0
0,5	0,8	0,1	507	5,65	0
0,5	0,8	0,01	36	5,65	0
0,6	0,6	0,1	17	5,65	0
0,6	0,6	0,01	636	5,65	0
0,6	0,7	0,1	93	5,65	0
0,6	0,7	0,01	11	5,65	0
0,6	0,8	0,1	90	5,65	0
0,6	0,8	0,01	349	5,65	0

Na Tabela 5.10, temos os resultados obtidos para o exemplo 2 Caso *a*. Este é um exemplo com simetria diedral cuja massa da solução ótima é de 163,47 Kg. Neste caso, a variação dos parâmetros levou a uma variação na massa final encontrada dentro das 2000 iterações. Observa-se ainda, que em nenhum dos casos analisados o algoritmo encontrou a melhor solução, que é de 163,47 Kg.

Tabela 5.10: Calibração de parâmetros: Exemplo 2 Caso a.

Pi	Pc	Pm	Iteração Ótimo	Massa Final	Penalização
0,4	0,6	0,1	1203	426,96	0
0,4	0,6	0,01	1320	326,81	72,3272
0,4	0,7	0,1	1219	462,11	0
0,4	0,7	0,01	1553	337,93	0
0,4	0,8	0,1	1538	396,23	0
0,4	0,8	0,01	1973	304,20	7,15176
0,5	0,6	0,1	1281	391,80	0
0,5	0,6	0,01	1974	293,79	0
0,5	0,7	0,1	1996	453,65	0
0,5	0,7	0,01	1810	322,79	0
0,5	0,8	0,1	747	416,82	0
0,5	0,8	0,01	1788	265,91	0
0,6	0,6	0,1	810	438,55	0
0,6	0,6	0,01	754	281,51	0
0,6	0,7	0,1	1692	416,82	0
0,6	0,7	0,01	1664	251,35	0
0,6	0,8	0,1	937	428,25	0
0,6	0,8	0,01	1774	334,92	0

Os resultados obtidos para o exemplo 2 Caso *b* estão na Tabela 5.11. Neste caso aproveitou-se a simetria diedral, e o exemplo convergiu para o resultado de menor massa em todos os conjuntos de parâmetros testados, variando-se apenas a iteração em que se chegou no melhor resultado. Neste caso, observa-se também uma grande variação no número de iterações para se obter a melhor solução.

Tabela 5.11: Calibração de parâmetros: Exemplo 2 Caso b.

Pi	Pc	Pm	Iteração Ótimo	Valor Final	Penalização
0,4	0,6	0,1	20	81,73	0
0,4	0,6	0,01	310	81,73	0
0,4	0,7	0,1	316	81,73	0
0,4	0,7	0,01	254	81,73	0
0,4	0,8	0,1	198	81,73	0
0,4	0,8	0,01	312	81,73	0
0,5	0,6	0,1	83	81,73	0
0,5	0,6	0,01	692	81,73	0
0,5	0,7	0,1	121	81,73	0
0,5	0,7	0,01	185	81,73	0
0,5	0,8	0,1	194	81,73	0
0,5	0,8	0,01	954	81,73	0
0,6	0,6	0,1	52	81,73	0
0,6	0,6	0,01	227	81,73	0
0,6	0,7	0,1	94	81,73	0
0,6	0,7	0,01	345	81,73	0
0,6	0,8	0,1	179	81,73	0
0,6	0,8	0,01	580	81,73	0

A Tabela 5.12, mostra os resultados obtidos para o exemplo 5. Para este exemplo observou-se que em alguns conjuntos de parâmetros as soluções obtidas eram não penalizadas, o que mostra uma maior tendência a se chegar na estrutura de massa mínima encontrada, que é de 7647,59 Kg.

Tabela 5.12: Calibração de parâmetros: Exemplo 3.

Pi	Pc	Pm	Iteração Ótimo	Valor Final	Penalização
0,4	0,6	0,1	557	11296,65	10859,39468
0,4	0,6	0,01	1238	12411,88	0
0,4	0,7	0,1	1079	11468,61	0
0,4	0,7	0,01	1985	11446,95	0
0,4	0,8	0,1	1174	11296,65	0
0,4	0,8	0,01	1370	11468,61	0
0,5	0,6	0,1	1504	9931,540	0
0,5	0,6	0,01	1746	13680,18	0
0,5	0,7	0,1	1908	11025,10	0
0,5	0,7	0,01	1127	10528,11	0
0,5	0,8	0,1	1985	11718,49	0
0,5	0,8	0,01	1766	12339,50	0
0,6	0,6	0,1	397	199900,04	198634,5109
0,6	0,6	0,01	49	199900,04	198634,5109
0,6	0,7	0,1	1666	11471,386	0
0,6	0,7	0,01	62	199900,04	198634,5109
0,6	0,8	0,1	320	199900,04	198634,5109
0,6	0,8	0,01	58	199900,04	198634,5109

Diante desses resultados é difícil estabelecer um conjunto ótimo de parâmetros para todos os exemplos, mas pode-se perceber que a combinação de parâmetros $P_m = 0.01$, $P_c = 0.7$ obteve quatro melhores resultados dos 5 testes realizados e $P_i = 0,6$ obteve dois dos melhores resultados entre os quatro anteriores.

A fim de ilustrar o estudo de calibração dos parâmetros, fez-se gráficos comparativos do desempenho dos exemplos ao longo das iterações para todos os conjuntos de parâmetros. Nos gráficos das figuras 5.10 e 5.11 são mostrados os desempenhos do exemplo 1 caso *a* (figura 5.10) e caso *b* (figura 5.11),utilizando-se os parâmetros $P_m = 0.01$, $P_c = 0.7$ e variando-se P_i.

Para todos os casos observa-se uma queda sistemática do valor da função de fitness ao longo das iterações. No caso do exemplo 1 caso *a*, apenas uma evolução ($P_i = 0,5$) não conduziu à estrutura de menor massa encontrada, antes de 500 iterações.

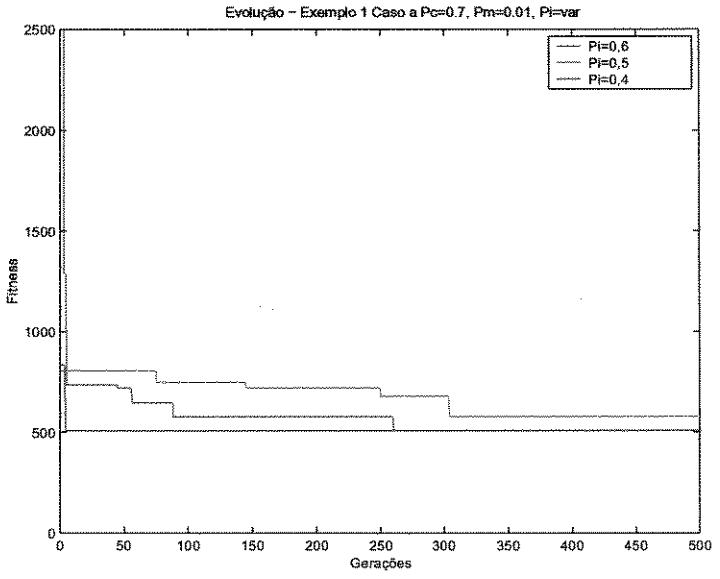


Figura 5.10: Gráfico comparativo - Exemplo 1 Caso a.

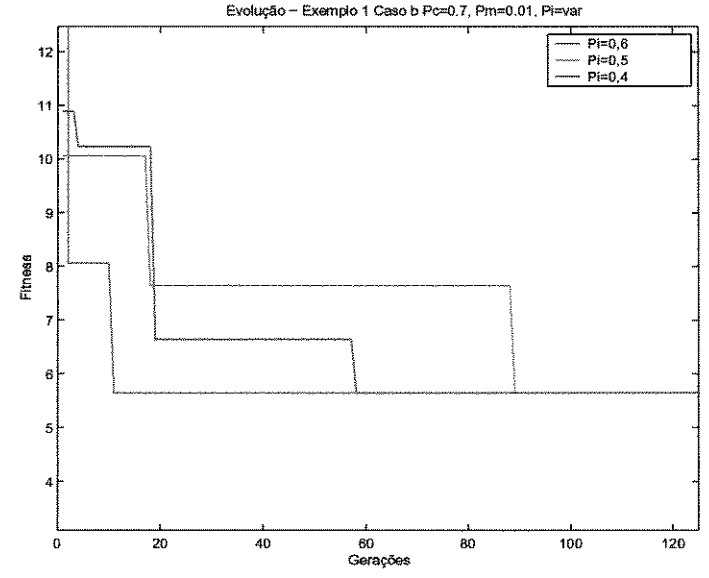


Figura 5.11: Gráfico comparativo - Exemplo 1 Caso b.

Observa-se neste caso, que a única diferença entre os dois exemplos é a rigidez de cada barra. É importante observar que a rigidez das barras varia diretamente com a área da seção transversal da barra e inversamente como o comprimento da barra.

Assim, as configurações ótimas com menor rigidez foram penalizadas devido a altos valores de deslocamento e tensão, não tendo sido selecionadas. Este fato mostra a importância da definição dos limites das restrições no problema.

De uma maneira global, o acompanhamento da evolução da função de fitness é mais uma ferramenta útil para verificar a sensibilidade dos parâmetros sobre a solução final encontrada.

5.4 Resultados Numéricos

5.4.1 Estruturas Reticuladas

Os testes escolhidos encontram-se amplamente discutidos na literatura ([8] e [35]) e são clássicos na área de otimização. Esses exemplos têm como objetivo validar o algoritmo e demonstrar sua eficiência e versatilidade. A descrição dos exemplos estão na seção 5.1 e suas características se encontram nas Tabelas 5.1, 5.2 e 5.3.

O objetivo do exemplo 1 caso *a* é validar o algoritmo implementado através da observação do resultado obtido pelo Algoritmo Genético implementado com o resultado obtido pelo algoritmo encontrado em [8]. Nesse artigo o autor utiliza um Algoritmo Genético com codificação real, onde varia-se a seção transversal da barra, dado que determina a existência ou não da barra na treliça.

Como melhor resultado para este exemplo obteve-se a treliça mostrada na figura 5.12, adotando-se 2000 gerações e população de 20 indivíduos.

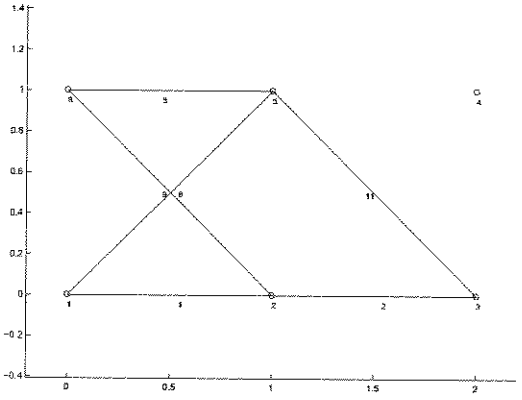


Figura 5.12: Resultado obtido para o Exemplo 1 Caso a.

O resultado obtido neste exemplo tem o mesmo layout que o resultado obtido em [8]. Vale ressaltar que o algoritmo utilizado em [8] utiliza uma abordagem diferente, o que não permite a comparação direta da massa da estrutura final. O resultado é

bastante satisfatório, pois chegou-se à uma configuração muito boa em um número pequeno de iterações e uma população relativamente pequena, visto que [8] utilizou populações de tamanho 300 e 450 indivíduos.

A treliça utilizada no exemplo 1 caso *b* é muito semelhante à do Exemplo1 caso *a*, porém as dimensões são diferentes. Este exemplo pretende mostrar a influência das dimensões das barras da treliça no resultado final.

Como melhor resultado para este exemplo obteve-se a treliça mostrada na figura 5.13, adotando-se 2000 gerações e população de 20 indivíduos.

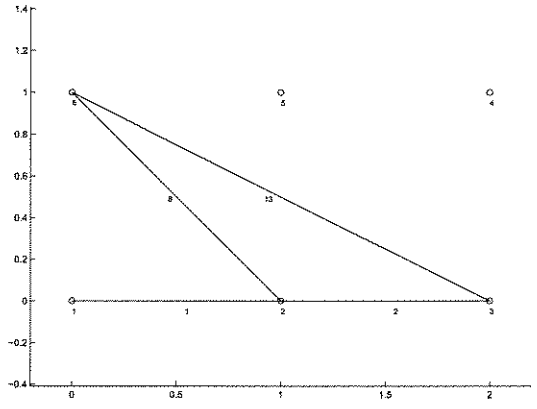


Figura 5.13: Resultado obtido para o Exemplo 1 Caso b.

Observa-se que devido às características da estrutura utilizada neste exemplo foi possível conseguir uma treliça com um número menor de barras, que atende plenamente às condições de contorno impostas. Com este resultado pode-se considerar que o desempenho do algoritmo é confiável. Observa-se também a influência da rigidez das barras no processo, assim como dos limites das restrições estabelecidos. No exemplo 1 caso *a*, as soluções com 4 barras foram penalizadas ao longo das iterações fato que levou a um resultado diferente.

Através do exemplo 2 caso *a* quer-se analisar o desempenho do método para uma estrutura um pouco maior, aumentando a complexidade do problema.

Como melhor resultado para este exemplo obteve-se a treliça mostrada na figura 5.14, adotando-se 2000 gerações e população de 20 indivíduos.

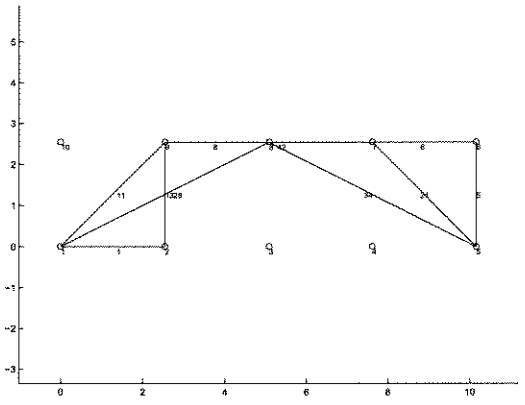


Figura 5.14: Resultado obtido para o Exemplo 2 Caso a.

Pode-se observar que o algoritmo está conduzindo a uma solução satisfatória, pois há apenas dois triângulos além do layout esperado. Desta forma espera-se que, com um número maior de gerações ou uma população maior, o algoritmo virá a eliminá-los. Ao observar-se este resultado depara-se com um problema conceitual: a falta de simetria. Como a estrutura original é simétrica espera-se uma solução também simétrica.

Aumentando o tamanho da população para 40 indivíduos e evoluindo o programa até 2000 gerações obtém-se o resultado esperado, que é mostrado na figura 5.15.

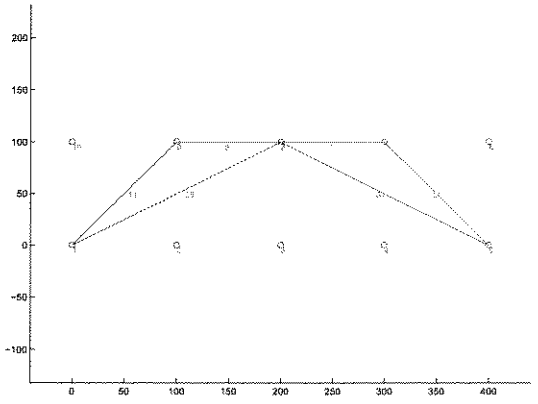


Figura 5.15: Resultado simétrico obtido para o Exemplo 2 Caso a.

Frente ao problema acima, utiliza-se a simetrização do exemplo 2 caso a, isto é, representa-se apenas metade da estrutura, e impõe-se as condições de simetria conforme mostrado na Tabela 5.2. Logo, a solução do exemplo 2 caso b é a metade da solução do exemplo 2 caso a.

Como melhor resultado obteve-se a treliça mostrada na figura 5.16.

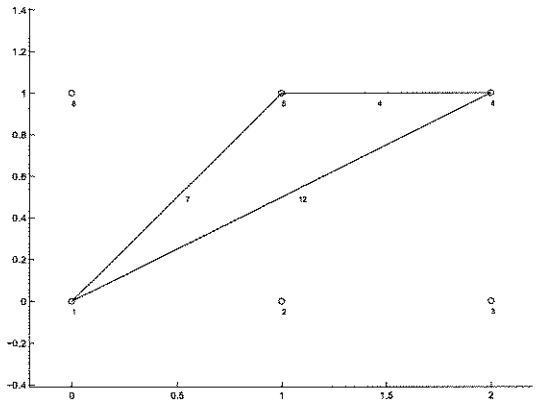


Figura 5.16: Resultado obtido para o Exemplo 2 Caso b.

Com a redução do problema e considerando a simetria a priori, o algoritmo convergiu rapidamente ao resultado, eliminando os problemas do modelo anterior. Isto nos leva a concluir que quando os problemas são simétricos pode-se impor as condições de simetria desde o início da modelagem a fim de conseguir uma convergência mais rápida. O algoritmo por si só não possui ferramentas para identificar estas propriedades. A figura 5.17 mostra a estrutura completa obtida pelo exemplo 2 caso b.

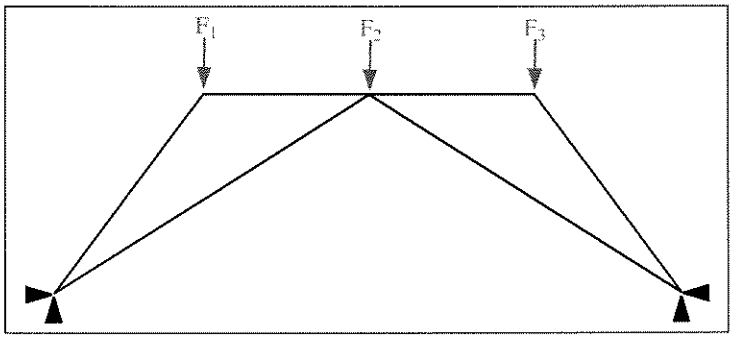


Figura 5.17: Estrutura completa do Exemplo 2 Caso b.

O exemplo 3 caso *a* refere-se a uma treliça em duas camadas e o objetivo é aplicar o algoritmo ao caso de estruturas mais complexas, isto é com dois níveis de barras, e com um espaço inicial de trabalho bem maior.

Como resultado para este exemplo, usando-se 2000 gerações e 40 indivíduos por população, obteve-se a treliça mostrada na figura 5.18. Neste caso o layout encontrado mostra uma tendência à concentração das barras nas laterais da estrutura,

com maior concentração de barras na primeira camada da estrutura. Observa-se também a não simetria do sistema.

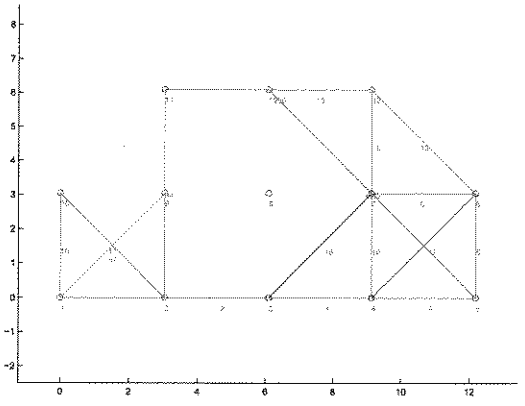


Figura 5.18: Resultado para o Exemplo 3 Caso a.

A simetrização do problema remete ao exemplo 3 caso *b*. O resultado deste problema é a treliça mostrada na figura 5.19.

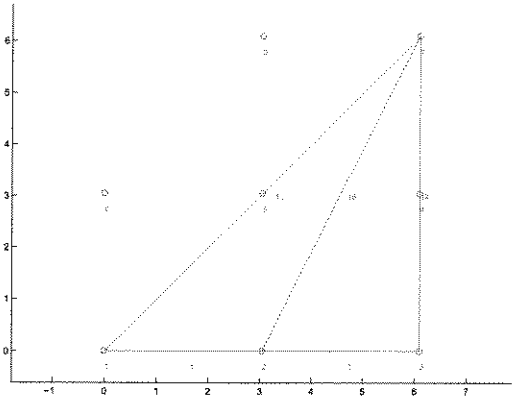


Figura 5.19: Resultado para o Exemplo 3 Caso b.

O algoritmo convergiu rapidamente ao resultado, eliminando os problemas do modelo não simétrico. A estrutura completa obtida pelo exemplo 3 caso b. é mostrada na figura 5.20.

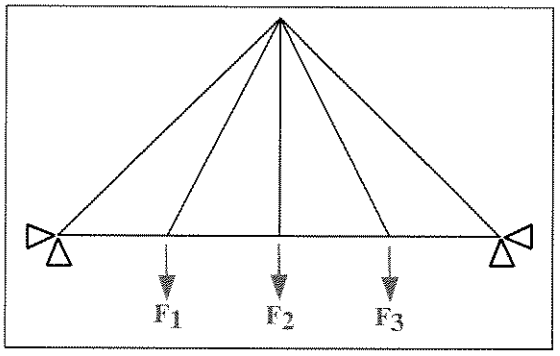


Figura 5.20: Estrutura completa Exemplo 3 Caso b.

5.4.2 Problemas em estado plano de tensões

Os exemplos escolhidos para esta categoria também são exemplos clássicos da literatura e que podem ser encontrados em [29]. O objetivo destes exemplos é observar o comportamento do algoritmo para este tipo de problema e verificar se eles podem ser resolvidos pelo mesmo tipo de método, isto é, o algoritmo binário com *crossover* uniforme. Em todos os casos adota-se 2000 gerações e as populações são de 40 indivíduos.

O exemplo 4 apresenta a estrutura de Duas Barras, que foi escolhida por possuir solução conhecida, mostrada na Figura 5.21.

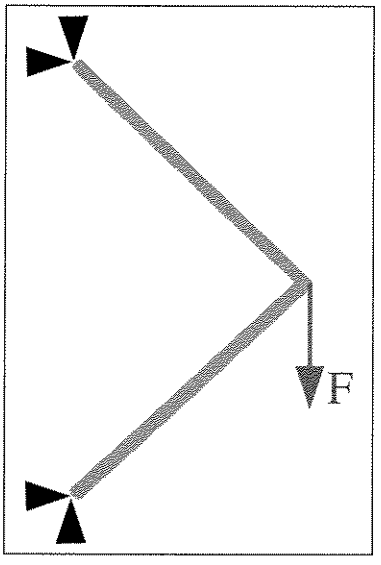


Figura 5.21: Solução analítica para a estrutura de Duas Barras.

Neste caso parte-se de um espaço inicial de trabalho que engloba toda a área, com $L_x = 10m$ e $L_Y = 24m$. Adotou-se elementos quadrilaterais de quatro nós, em uma malha inicial de 240 elementos.

As soluções encontradas para este exemplo não foram satisfatórias, pois não mostraram tendência ao resultado conhecido, conforme pode ser visto na Figura 5.22. Algumas alterações nos parâmetros e nas malhas foram testadas, porém não houve nenhum progresso significativo. Não encontrou-se muitos casos na literatura que resolvem este problema usando Algoritmos Genéticose os artigos encontrados não forneceram dicas suficientes para melhorar o nosso algoritmo.

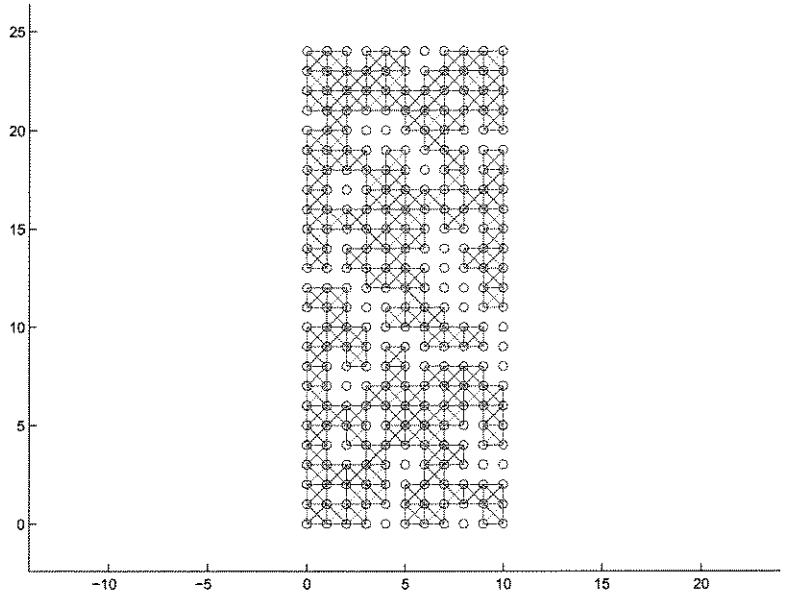


Figura 5.22: Resultado obtido para a estrutura de Duas Barras.

O exemplo 5 apresenta a estrutura de Michell [29], que foi escolhida por ser um exemplo clássico e por possuir solução analítica, que é apresentada na Figura 5.23. Neste exemplo trabalha-se com metade da estrutura, isto é, simetriza-se o problema.

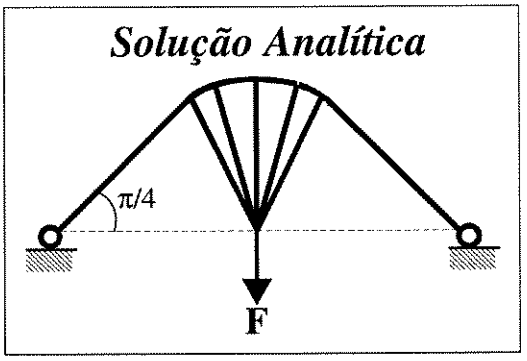


Figura 5.23: Solução analítica para a estrutura de Michell.

Este exemplo foi resolvido primeiramente com uma malha grosseira, o que não permite uma solução precisa.

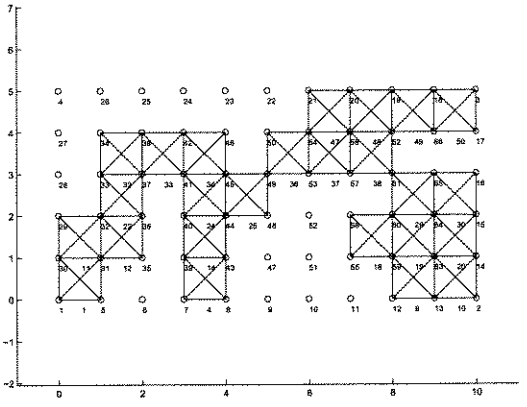


Figura 5.24: Resultado obtido para a estrutura de Michell com malha menos refinada.

Observando-se esses resultados percebe-se a necessidade em se refinar a malha para que se obtenha melhores resultados. Além do refinamento da malha, é necessário ampliar os estudos para o caso de estado plano de tensões, procurando-se algoritmos alternativos, que sejam mais eficientes para este problema. Os estudos aqui apresentados, permitem concluir que o algoritmo binário na forma como foi implementado, não se mostrou satisfatório para a solução de problemas de otimização topológica usando-se modelos de meios contínuos elásticos.

A possibilidade de um refinamento maior das malhas, pode levar a soluções viáveis, todavia esta solução implica em um alto custo computacional, por isso não foi aprofundada neste trabalho.

Capítulo 6

Conclusões

Neste capítulo são apresentadas as principais conclusões a respeito do trabalho desenvolvido, bem como as sugestões para a continuidade da pesquisa em questão.

Neste trabalho foi proposto o uso de um Algoritmo Genético para a otimização topológica de estruturas. O método proposto foi um Algoritmo Genético binário, com o uso de *crossover* uniforme, mutação simples, seleção elitista e abordagem Pittsburg. Diferente da maioria dos artigos, decidiu-se trabalhar com Algoritmo Genético binário e não real, pois entendeu-se que para este tipo de otimização a representação binária seria suficiente, pois a representação indica presença ou ausência de material.

Os Algoritmos Genéticos são técnicas úteis e relativamente robustas para se resolver problemas complexos de otimização global, no qual os métodos baseados em gradiente nem sempre são eficientes uma vez que podem convergir para um ótimo local. De uma forma geral os Algoritmos Genéticos manipulam com eficiência seus operadores, permitindo freqüêntemente a obtenção de soluções interessantes sem custos computacionais elevados.

Devido ao grande número de variáveis, alguns problemas estruturais são de difícil solução, porém o grande potencial do método está na facilidade de escapar de mínimos locais.

Observou-se ao longo do trabalho que a formulação do problema é fundamental para o desempenho do método. É preciso uma boa formulação das restrições do problema a fim de diminuir o espaço de busca; como o Algoritmo Genético é uma técnica pseudo-aleatória, espaços de busca muito grande dificultam o bom andamento do processo de otimização, pois o número de pontos a serem testados se torna proibitivo.

O Algoritmo Genético em questão foi implementado com técnicas de programação orientada a objetos utilizando-se linguagem C++. O algoritmo tornou-se uma parte do módulo de otimização do programa *MefLab++*, que está sendo desenvolvido

por alunos e professores do Departamento de Mecânica Computacional da Faculdade de Engenharia Mecânica da Unicamp.

A programação orientada a objetos foi escolhida por ser um importante recurso de programação computacional, pois permite o desenvolvimento de programas que possam estar em constante atualização devido à facilidade de se acrescentar novas rotinas e módulos computacionais. Utilizou-se essa propriedade da linguagem para facilitar a verificação das restrições do Algoritmo Genético. Todo o cálculo pelo Método dos Elementos Finitos foi feito fora do módulo de otimização.

A primeira parte do trabalho foi a abstração dos objetos e classes que iriam compor o projeto do programa. Esta foi uma fase bastante complexa e muito importante para que se obtivesse um bom resultado final, pois a definição e a identificação das etapas, dos componentes e dos métodos necessários para a resolução do problema deu-se aqui. Nesta etapa tivemos o cuidado de direcionar o projeto para facilitar o desenvolvimento dos próximos módulos.

A segunda etapa foi a identificação de quais seriam as operações e as classes do *MefLab++* que seriam utilizadas no Algoritmo Genético. Além da identificação e do entendimento dessas operações e classes, foram necessárias algumas adaptações das operações já existentes e a inclusão de algumas novas operações para se suprir as necessidades do Algoritmo Genético. Esta etapa consumiu uma considerável parcela de tempo e dedicação do projeto.

A terceira etapa foi a implementação do Algoritmo Genético propriamente dito. Esta implementação é relativamente simples e foi feita com certa facilidade.

A elaboração dos exemplos e de seus respectivos arquivos de entrada também exigiu cuidados especiais, tendo em vista o número de dados que podem ser manipulados. Esta elaboração contou com a ajuda do software ANSYS, onde foi feito todo o pré-processamento, através do uso de macro comandos para a geração dos arquivos de entrada do *MefLab++*. .

Vale ressaltar algumas dificuldades encontradas ao longo do desenvolvimento do projeto. Uma delas se deu na verificação da singularidade da matriz de rigidez dos elementos. Primeiramente utilizou-se a *Decomposição de Cholesky*, que é tradicionalmente usada para se verificar se uma matriz é definida positiva ou não. Devido a erros numéricos da máquina, bem como à topologia de algumas matrizes geradas, não foi possível a recuperação da matriz original, a partir da matriz gerada pela Decomposição de Cholesky, impedindo a verificação da singularidade. A segunda tentativa foi a utilização da *Decomposição LU*, onde nos deparamos com a mesma dificuldade: erros numéricos impossibilitaram uma verificação direta da diagonal de L. Decidiu-se, então, verificar os autovalores da matriz de rigidez dos elementos. Esta técnica funcionou bem para os problemas com uma malha pequena, mas quando problemas com malhas maiores foram testados, o método tornou-se muito custoso.

Uma última tentativa ainda foi testada: a utilização do método dos sub-espaços juntamente a um shift nos autovalores [9]. Os resultados foram satisfatórios, mas o custo computacional aumentou pois a implementação do método de sub-espaço não foi realizada de forma eficiente. Outras possibilidades devem ser estudadas em trabalhos futuros.

Outra dificuldade foi estabelecer os limites de deslocamento e tensões para os problemas. Estes valores interferem diretamente no resultado final; portanto é preciso ter muito cuidado e usar da experiência para a escolha dos mesmos. Este fato reforça a idéia de que é difícil estabelecer métodos totalmente automáticos para otimização topológica de estruturas usando-se Algoritmo Genético. Isto é, a experiência, o conhecimento e a análise minuciosa dos resultados e da evolução do Algoritmo Genético são fundamentais para a obtenção de soluções viáveis, dependendo significativamente da experiência dos engenheiros e analistas.

Uma análise global dos resultados permite verificar que o algoritmo convergiu para soluções satisfatórias nos problemas de estruturas reticuladas, chegando na maioria das vezes em soluções ótimas ou em soluções perfeitamente aceitáveis do ponto de vista da engenharia.

Porém, é preciso alguns estudos para a aplicação nas estruturas em elasticidade plana, cujos resultados não se mostraram eficientes. A utilização de operadores genéticos especiais para este tipo de problema é um dos pontos que devem ser estudados.

Alguns pontos fundamentais podem ser apontados como assunto de posteriores estudos. A hibridação do Algoritmo Genético, isto é, a combinação de outros métodos ao Algoritmo Genético, em especial a busca local, podem levar ao aceleramento da convergência e à melhora das soluções encontradas. A utilização de cromossomos reais ou mistos também podem ajudar na melhora do algoritmo. Quanto aos aspectos da formulação, pode-se utilizar a otimização topológica juntamente com a paramétrica e a de forma, afim de se conseguir melhores resultados.

Referências Bibliográficas

[1] J. S. ARORA. *Introduction to Optimun Design.* Mc Graw-Hill, 1989.

[2] M. S. BAZARAA, J. J. JARVIS, and H. D. SHERALI. *Linear Programing and Network Flows - second Edition.* John Wiley & Sons, 1990.

[3] E. B. BECKER, G. F. CAREY, and J. T. ODEN. *Finite Elements - An Introdution - Volume I.* Prentice-Hall, 1981.

[4] F.P. BEER and E.R. JOHNSTON JR. *Mecânica Vetorial para Engenheiros - Estática.* Makron Books do Brasil, 5ª Edição, 1994.

[5] C. G. BRAGA. O uso de Algoritmos Genéticos para aplicação em problemas de otimização de sistemas mecânicos. Master's thesis, Universidade Federal de Uberlândia, Centro de Ciências Exatas e Tecnologia, 1998.

[6] W. F. CHEN and A. F. SALEEB. *Constitutive Equation for Engineering Materials - Elasticity and Modeling - Volume I.* John Wiley & Sons, 1982.

[7] R. D. COOK, D. S. MALKUS, and M. E. PLESHA. *Concepts and Applications of Finite Elements Analysis - Third Edition.* John Wiley & Sons, 1989.

[8] K. DEB and S. GULATI. Design of truss-structures for minimum weight using genetic algorithms. *Finite Elements in Analisys and Design.*, 37:447–465, 2001.

[9] G. DHATT and G. TOUZOT. *The Finite Element Method Displayed.* John Wiley & Sons, 1984.

[10] M.E.M. EL-SAYED and C.-K. HSIUNG. Optimun structural design with parallel finite element analysis. *Computers & Structures*, 40:1469–1474, 1991.

[11] M.S. ELDERED, W.E. HART, W.J. BOHNHOFF, V.J. ROMERO, S.A. HUTCHINSON, and A.G. SALINGER. Utilinzing object-oriented design to build advanced optimization strategies with generic implementation. *Proc. 6^{th} AIAA/USAF/NASA/ISSMO Symp on Multidiciplinary Analyses and Optimization.*, pages 1568–1582, 1996.

[12] J.V. FERREIRA. *Dynamic reponse analisys of structures with nonlinear components.* PhD thesis, Imperial College of Science, Tecnology and Medicine, University of London, Department of Mechanical Engeneering., 1998.

[13] A. FRIEDLANDER. *Elementos de Programação Não-linear.* Editora da Unicamp, 1994.

[14] D.E. GOLDBERG. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989.

[15] R.T. HAFTKA and M. P. KAMAT. *Elements of Structural Optimization.* Hardbound, 1984.

[16] J. HOLLAND. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[17] C. KANE, F. JOUVE, and SCHOENAUER M. Structural topology optimization in linear and nonlinear elasticity using genetic algorithms. *21st ASME Design Automatic Conference, Boston MA*, 1995.

[18] H. KAWAMURA, H. OHMORI, and N. KITO. Truss topology optimization by a modified genetic algorithm. *Structure and Multidiciplinary Optimization*, 23:467–472, 2002.

[19] C.S. KRISHNAMOORTHY, P. PRASANNA VENKATESH, and R. SUDARSHAN. Object-oriented framework for genetic algorithms with application to space truss optimization. *Journal of Computing in civil engineering*, 16, 2002.

[20] N.D. LAGAROS, M. PAPADRAKAKIS, and G. KOKOSSALAKIS. Structural optimization using evolutionary algorithms. *Computers and Structures*, 80:571–589, 2002.

[21] J. LIBERTY. *Aprenda em 24 horas C++ - Tradução: Ana Maria Netto Guz.* Campus, 1998.

[22] D. G. LUENBERGER. *LINEAR AND NONLINEAR PROGRAMMING - Second Edition.* Addison-Wesley Publishing Company, 1984.

[23] Z. MICHALEWICZ. *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag, 3rd edition, 1996.

[24] H.M. NUSSENZVEIG. *Curso de Física Básica - Mecânica.* Editora Edgard Blucher Ltda., 2ª Edição, 1981.

[25] R. PAVANELLO. Introdução ao Método dos Elementos Finitos. Notas de Aula, Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica, Departamento de Mecânica Computacional, 1997.

[26] F. PIZZIRANI. Matrizes definidas positivas e problemas que as envolvem. Relatório de Iniciação Científica - FAPESP Processo número 99/03033-8, São Paulo: Instituto de Matemática Estatística e Computação Científica, Universidade Estadual de Campinas, 2000.

[27] R.G.L. RICHE, C. KNOPF-LENOIR, and R.T. HAFTKA. A segregate genetic algorithm for constrained structural optmization. *In: L. J. ESHELMAN (Eds). Proceedings of the 6th International Conference on Genetic Algorithm.*, pages 558–565, 1995.

[28] H. M. SANT´ANNA. Truss optimization in aerospace structures. *Socity of Automotive Engineers.*, 2001.

[29] J. A. B. SILVA. Investigação de um método evolucionário de otimização estrutural. Master's thesis, Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica, 2001.

[30] K.R. SYMON. *Mecânica.* Editora Campus, 6ª Edição, 1982.

[31] Y. TANIMURA, HIROYASU T., and M. MIKI. Discussion on distributed genetic algorithms for designing truss structures. *The fifth International Conference and Exihibition on Hight-Performance Computing in the Asia-Pacific Region*, 9:1–13, 2001.

[32] G. N. VANDERPLAATS. *Numerical Optimization Techniques for Engineering Design.* 1984, Mc Graw-Hill.

[33] G.N. VANDERPLAATS. Thirty years of modern strructural optimization. *Advances in Engeneering Software*, 16:81–88, 1993.

[34] F. VON ZUBEN. Computação evolutiva. Notas de Aula, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica, Departamento de Controle e Automação, 2001.

[35] T. YOKOTA, T. TAGUCHI, and M. GEN. A solution method for optimal weight design problem of 10 bar truss using genetic algorithms. *23rd International Conference on Computers and Industrial Engineering*, 1-2:367–372, 1998.

[36] O.C. ZIENKIEWICZ. *The Finite Element Method in Engineering Science - 4^{th} Edition.*

Apêndice A

A Teoria dos Esquemas

A *Teoria do Esquemas* foi proposta por Holland para explicar porque os Algoritmos Genéticos funcionam. Esta teoria é válida apenas para codificações binárias, e tem como principais resultados o *Teorema do crescimento dos esquemas* e a *hipótese dos blocos construtivos* [34].

Um esquema é uma representação capaz de descrever diversos cromossomos simultaneamente. Um esquema é construído inserindo-se um caractere *"don't care"(*)* no alfabeto dos genes, indicando que aquele gene representa qualquer alelo.

Como exemplo tem-se o esquema $[1 * 01001]$ que representa os cromossomos [1001001] e [1101001]. Cada caractere $*$ fornece duas possibilidades para o gene: 1 ou 0. O esquema $[* * * * * * *]$ representa todos os cromossomos de tamanho 7.

Cada esquema representa 2^r cromossomos, onde r é o número de caracteres $*$ presentes no esquema. Por outro lado, cada cromossomo de comprimento m é representado por 2^m esquemas. Logo, considerando-se cromossomos de comprimento m, há um total de 3^m possíveis esquemas. Numa população de tamanho n há entre 2^m e $n2^n$ possíveis diferentes esquemas.

Alguns termos utilizados devem ser definidos. São eles:

- A ordem $o(S)$ de um esquema S, é definida como o número de zeros e uns presentes no esquema, isto é, o número de posições fixas presentes no esquema.
- O comprimento definitório de um esquema S, denotado por $\delta(S)$, é a maior distância entre posições fixas de um cromossomo. O comprimento definitório define o nível de compactação da informação contida no esquema.
- O fitness de um esquema S na geração t, $eval(S,t)$, é definido como a média dos fitness de todos os cromossomos na população representados pelo esquema S.

Assumindo-se que há p cromossomos $\{x_{i_1}^t, \ldots, x_{i_p}^t\}$ representados pelo esquema S_i na geração t, temos:

$$eval(S_i, t) = \frac{1}{p} \sum_{j=1}^{p} (eval x_{i_j}^t) \tag{A.1}$$

onde $eval(x_{i_j}^t)$ é o fitness do indivíduo $x_{i_1}^t$.

Seja *tam_pop* o tamanho da população. O fitness médio,$\bar{F}(t)$, da população na geração t, é dado por:

$$\bar{F}(t) = \frac{1}{tam\_pop} \sum_{i=1}^{tam\_pop} eval(x_i^t) \tag{A.2}$$

Sejam p_c e p_m as probabilidades de crossover e mutação, respectivamente, e m o comprimento dos cromossomos. Seja $\xi(S_i, t)$ o número de cromossomos representados pelo esquema S_i na geração t. Pode-se mostrar que [23]:

$$\xi(S_i, t+1) \geq \frac{\xi(S_i, t) eval(S_i, t)}{\bar{F}(t)} \left[1 - p_c \frac{\delta(S_i)}{m-1} - o(S_i) p_m\right] \tag{A.3}$$

A equação A.3 é a equação de crescimento reprodutivo do esquema. Ela é deduzida supondo-se que a função de fitness assume apenas valores positivos.

A equação A.3 mostra que a seleção aumenta a amostragem de esquemas cujos fitness está acima da média da população, e este aumento é exponencial. A seleção, por si só, não introduz nenhum novo esquema. Esta é a razão da introdução do operador de crossover: possibilitar a troca de informação estruturada, ainda que aleatória. Além disso, o operador de mutação introduz uma variabilidade maior na população. O efeito (destrutivo) combinado destes operadores não é significativo se o esquema é curto e de ordem baixa.

Desta forma, o teorema dos esquemas pode ser enunciado como:

Teorema dos Esquemas: *Esquemas com comprimento definitório curto, de ordem baixa, e com fitness acima da média, têm um aumento exponencial de sua participação em gerações consecutivas de uma Algoritmo Genético.*

A prova deste teorema se encontra em [16].

Uma conseqüência imediata deste teorema é que os Algoritmos Genéticos tendem a explorar o espaço por meio de esquemas curtos e de baixa ordem que, subseqüentemente, são usados para troca de informação durante o crossover.

Hipótese dos Blocos Construtivos: Um Algoritmo Genético busca desempenho quase-ótimo através da justaposição de esquemas curtos, de baixa ordem e alto desempenho, chamados *blocos construtivos.*

Em uma população de tamanho *tam_pop*, indivíduos de comprimento m processam pelo menos 2^m e no máximo 2^{tam_pop} esquemas. Alguns deles são processados de forma útil: são amostrados a uma taxa crescente exponencial; e outros são quebrados por meio de crossover e mutação.

Holland mostrou que, em uma população de tamanho tam_pop, pelo menos tam_pop^3 são processados de forma útil. Esta propriedade foi denominada *paralelismo implícito*, pois é obtida sem nenhuma exigência extra de memória e processamento.

Note que , em alguns problemas, alguns blocos construtivos podem direcionar erroneamente o algoritmo, levando-o a convergir a pontos sub-ótimos. Este fenômeno é conhecido como decepção.

Assim, a hipótese dos blocos construtivos não fornece uma explicação definitiva do porquê os Algoritmos Genéticos funcionam, mas é apenas uma indicação do porquê os Algoritmos Genéticos funcionam para uma certa classe de problemas.

Esta é uma informação que pode ser muito explorada, porém não fará parte destes estudos, ficando para trabalhos mais aprofundados neste assunto.

Apêndice B

Exemplo de um Arquivo de Entrada do Programa *MefLab++*

Os aquivos texto que contém os dados de entrada do programa *MefLab++* são organizados da maneira como o apresentado abaixo:

```
titulo = [optimization]
Task = [
    EvolutionaryOptimization = [
        GeneticAlgorithm = [
            BinaryGA =[
                (num_ger)     (100)
                (prob_cross)  (0.8)
                (prob_mut)    (0.1)
                (size_pop)    (20)
                (max_bar)     (13)
                            ]
        Fitness = [
                Constraints = [
                        BasicNodes = [
                                (NodeList) (1 2 3 6)
                                (weight) (1000)    ]
                        Singularity = [
                                (weight) (1000)
                                ]
                        DisplacementLimit = [
                                (max_disp) (0.01)
                                (weight) (10)
```

```
                                                ]
                                        StressLimit = [
                                                (max_stress) (100000000)
                                                (weight) (10)
                                                ]
                                        ]
                                ObjectiveFunction = [
                                        TotalMass = [
                                                ( ) ( )
                                                ]
                                        ]
                                ]
                        ]
                ]
        ]
DomainProblem = [A]
Domain = [
        DomainList = [  ]
        Name = [A]
        DifferentialOperator = [Truss]
        SpaceDimension = [2]
        AproximationMethod = [FEM]
        MaterialBehaviour = [GHooke]
        BCList = [
                Displacement = [
                        (1) (dx) (0.0)
                        (1) (dy) (0.0)
                        (6) (dx) (0.0)
                        (6) (dy) (0.0)
                        ]
                Force = [
                        (2) (dy) (-100000.0)
                        (3) (dy) (-100000.0)
                        ]
                ]
        GeometricElements = [
                        Attributes = [(Linengeo) (Trussphy) (Steel) (Barra1) ]
                        Conectivity = [
                                ( 1) (1 2)
```

```
                    ( 2) (2 3)
                    ( 3) (3 4)
                    ( 4) (4 5)
                    ( 5) (5 6)
                    ( 6) (1 6)
                    ( 7) (2 5)
                    ( 8) (1 5)
                    ( 9) (2 6)
                    (10) (2 4)
                    (11) (3 5)
                    (12) (1 4)
                    (13) (3 6)
                    ]
          ]
GeometricNodes = [
                    (1) ( 0.00 0.00 0.00 )
                    (2) ( 3.60 0.00 0.00 )
                    (3) ( 7.20 0.00 0.00 )
                    (4) ( 7.20 3.60 0.00 )
                    (5) ( 3.60 3.60 0.00 )
                    (6) ( 0.00 3.60 0.00 )
                    ]
ElementTypeList = [
                    Linengeo = [(InterpolationOrder) (1) ]
                    trussphy = [(InterpolationOrder) (1) ]
                    ]
GeometricPropertiesList = [
                    Barra1 = [
                              (Area) (0.0025)
                              ]
                    ]
MaterialPropertiesList = [
          Steel = [
                    (E) (206700000000)
                    (Rho) (7800)
                    (Mi) (0.3)
                    ]
          ]
    ]
```

Apêndice C

Principais Funções do Código Fonte

Neste apêndice serão apresentados os códigos-fonte das funções mais importantes do programa. Essas funções fazem parte dos arquivos *fitness.cpp* e *Gntclgrt.cpp*.

```
void GeneticAlgorithm::give_CurrentWordInterpretation(Dictionary&
theDictionary,GeneticAlgorithm*& pGeneticAlgorithm){
  int wordcode;
  enum dictwords {wBinaryGA=1,wIntegerGA,wRealGA,wFitness};
  theDictionary.add_Word("BinaryGA",wBinaryGA);
  theDictionary.add_Word("IntegerGA",wIntegerGA);
  theDictionary.add_Word("RealGA",wRealGA);
  theDictionary.add_Word("Fitness",wFitness);

  while(!theDictionary.check_IfDescriptionHasFinished
  ("GeneticAlgorithm")){theDictionary.give_NextDictionaryWordCode
  (wordcode);
      switch (wordcode){
      case wBinaryGA:
         BinaryGA * pBinaryGA;
         pBinaryGA = new BinaryGA;
         pBinaryGA->give_CurrentWordInterpretation(theDictionary);
         pGeneticAlgorithm = pBinaryGA;
        break;
        case wIntegerGA:
         IntegerGA * pIntegerGA;
         pIntegerGA = new IntegerGA;
         pIntegerGA->give_CurrentWordInterpretation(theDictionary);
         pGeneticAlgorithm = pIntegerGA;
        break;
        case wRealGA:
         RealGA * pRealGA;
         pRealGA = new RealGA;
         pRealGA->give_CurrentWordInterpretation(theDictionary);
         pGeneticAlgorithm = pRealGA;
        break;
        case wFitness:
         pFitness = new Fitness;
         pFitness->give_CurrentWordInterpretation(theDictionary);
         pGeneticAlgorithm->set_pFitness(pFitness);
         break;
         default:
         cout << " The class " << wordcode << " is invalid data for
                                              Genetic Algorithm !!!! "
          << endl;
         break;
      }
   }
}
```

```
void GeneticAlgorithm::sort(ColumnVector grade,ColumnVector &indexes){
  /* Algorítmo para ordenar um vetor através de um vetor de índices
    chamado indexes */

   int i, j, p1, p2, aux;
   int size=grade.Nrows();

   // inicializa o vetor de índices
   for (i=1; i <= size; i++)
       indexes(i) = i;

   // executa o selection sort
   for (i=1; i <= size; i++){
      p1 = indexes(i);
      for (j=size; j >= i+1; j--){
         p2 = indexes(j);
         if(grade(p1) > grade(p2)){
            aux = indexes(i);
            indexes(i) = indexes(j);
            indexes(j) = aux;
            p1 = indexes(i);
         }
      }
   }
}
```

```
void BinaryGA::crossover(ColumnVector& pai1, ColumnVector& pai2){
/* Realiza crossover uniforme entre dois indivíduos selecionados por
roulette-wheel com probabilidade " alfa". Esta função recebe como
parâmetros os indivíduos pais e retorna os filhos gerados por cross-
over, através de referência.*/
    int aux,i;
    // Percorre o cromossomo
    for (i=1; i<=max_bar ; i++){
        //Realiza o crossover se rand menor que alfa
        if ((double) rand()/RAND_MAX < prob_cross ){
            aux=pai1(i);
            pai1(i)=pai2(i);        // Filho 1
            pai2(i)=aux;            // Filho 2
        }
    }
}
```

```
void BinaryGA::mutation(Matrix& pop, int &mut, Matrix &pop_mut){
/*Realiza mutação simples no indivíduo selecionado por roulette-wheel
com probabilidade "beta". Esta função recebe como parâmetros a popu-
lação,e o valor mut. Ela retorna a sub-população de indivíduos mutados
(pop_mut).*/

    int i, ind_selec,j;
    double aleat;

    // Preenche a população de mutação
    for (i=1; i<=mut; i++){
       aleat=ceil((double) rand()/RAND_MAX * size_pop);

       // Seleciona indivíduo aleatóriamente
       ind_selec=__max(1,aleat);

       for (j=1; j<=max_bar ; j++)

          // Realização da mutação
          if ((double) rand()/RAND_MAX < prob_mut){
                pop_mut(i,j)=1-pop(ind_selec,j);}
          else{
                // Recupera o indivíduo selecionado por roulette-wheel
                pop_mut(i,j)=pop(ind_selec,j);
          }
    }
}
```

```
int BinaryGA::selection(ColumnVector  grade){
/*  Esta função seleciona um indivíduo para sofrer mutação ou realizar
crossover através de roulette-wheel. Ela recebe como parâmetro o vetor
do fitness, e retorna a posição do indivíduo selecionado.*/

   int cont=1,i;    // Variável que identifica o indivíduo selecionado

   // Calcula um número aleatório entre 0 e total
   double aleat=(double) rand()/RAND_MAX;
   double max_grade=grade.MaximumAbsoluteValue();
   ColumnVector fit(size_pop);

   // Calcula a soma dos fitness normalizados dos indivíduos da população
   double total=grade.Sum();
   ColumnVector prob(size_pop);

   for (i=1; i<=size_pop; i++)
      prob(i)=grade(i)/total;

   // Zera a variável utilizada para selecionar o indivíduo
   double soma_parcial=prob(1);

   // procura o indivíduo a ser selecionado
   while (soma_parcial < aleat && cont < size_pop){
      cont++;
      soma_parcial=soma_parcial+prob(cont);
   }
   return cont;
}
```

```
void BinaryGA::initial_pop(Matrix &pop){
/* Geração da População Inicial aleatóriamente. Recebe como parâmetro
a população através de referência.*/

    int i,j;
    pop.ReDimension(size_pop,max_bar+2);

    //atualiza a semente a ser utilizada na geração de n. aleatórios
    srand( (unsigned)time( NULL ) );

    for (i=1; i<=max_bar; i=i+1)
        pop(1,i)=1;

    for (i=2 ; i<=size_pop ; i++){
        for (j=1; j<=max_bar ; j++){
            if (  (double)rand()/RAND_MAX < 0.5){
                // Atribui 0 se valor aleatório menor que o determinado.
                pop(i,j)=0; }
            else{
                // Atribui 1 se valor aleatório for maior
                pop(i,j)=1;
            }
        }
    }
}
```

```
void BinaryGA::apply_Method(Domain *pDomain){
// Programa que gerencia o Algoritmo Genético.

// Gera todos os dados do domínio para todos os elementos possíveis
pDomain->give_StaticData();

// Rotina que atribui os pesos às restrições dependente da massa total
da estrutura inicial
double mass=0;
pDomain->give_TotalMass(mass);

printf("%.18g \n",mass);
printf("   ");

Constraints *pConstraint=pFitness->get_pConstrains();

ConstraintList theConstList = pFitness->get_theConstraintList();

pConstraint = theConstList.give_FirstObject();
for (int i=1; i<=4; i++){
    switch (i)    {
        case 1:
            pConstraint->set_weight(10*mass);
            pConstraint = theConstList.give_NextObject();
            break;

        case 2:
            pConstraint->set_weight(8*mass);
            pConstraint = theConstList.give_NextObject();
            break;

        case 3:
            pConstraint->set_weight(mass);
            pConstraint = theConstList.give_NextObject();
            break;

        case 4:
            pConstraint->set_weight(mass);
            break;
   }
}
```

```
// Inicialização das variáveis
int elite=ceil(size_pop/4);    // Define o tamanho do conjunto pop_elite
int mut=ceil(size_pop/2);      // Define o tamanho da população com
                                  mutação e crossover
int ger, j;
int ind_pai1, ind_pai2;

//Inicialização das matrizes e vetores

// Inicializa a população de elite
Matrix pop_elite(elite, max_bar+2);

// Inicializa a população intermediária com aplicação de crossover
Matrix pop_cross(2*mut, max_bar+2);

// Inicializa a população intermediária com aplicação de mutação
Matrix pop_mut(mut,max_bar+2);
ColumnVector indexes(size_pop);
ColumnVector pai1(max_bar+2), pai2(max_bar+2);
ColumnVector fit(num_ger);
ColumnVector MassGer(num_ger);


initial_pop(pop);    // Gera a população inicial

//Calcula o fitness da população e armazena na última coluna de pop
pFitness->calc_fitness(pop,pDomain);

// Ordena os indivíduos pelo fitness
sort(pop.Column(max_bar+1), indexes);

for (ger=1; ger <= num_ger; ger++)  // Percorre as gerações
{
    // Gera a população de elite a partir da população total
    ( melhores indivíduos )
    give_pop_elite(pop_elite, elite, indexes);

    // Seleção dos indivíduos para crossover através de roulette-wheel
    for (j=1; j<=mut; j++)
    {
        ind_pai1=selection(pop.Column(max_bar+1));

        ind_pai2=ind_pai1;
```

```
  while (ind_pai1 == ind_pai2)
       // Garante que pai1 seja diferente de pai2
       ind_pai2=selection(pop.Column(max_bar+1));

  for (i=1; i<=max_bar; i++)
        {
            //Identifica o primeiro individuo selecionado (pai 1)
            pai1(i)=pop(ind_pai1,i);
            //Identifica o segundo individuo selecionado (pai 2)
            pai2(i)=pop(ind_pai2,i);
        }
   crossover(pai1,pai2);          // Realiza o crossover

        // Armazena os filhos na população de crossover
        for (i=1; i<=max_bar; i++)
        {
            pop_cross(j,i)=pai1(i);
            pop_cross(mut+j,i)=pai2(i);
        }
    }
    //Calcula o fitness da população de crossover e armazena na última
      coluna de pop_cross
    pFitness->calc_fitness(pop_cross,pDomain);

    mutation(pop_cross,mut,pop_mut);    // Realiza a mutação

    //Calcula o fitness da população de mutação e armazena na última
      coluna de pop_mut
    pFitness->calc_fitness(pop_mut,pDomain);

    //Define a nova população
    give_new_pop(pop, pop_elite, pop_cross, pop_mut, elite);
    // Ordena os indivíduos pelo fitness
    sort(pop.Column(max_bar+1), indexes);

    printf("%.18g %.18g %.18g %d \n",pop(indexes(1),max_bar+1)
    ,pop(indexes(1),max_bar+2),pop(indexes(1),max_bar+1)-
    pop(indexes(1),max_bar+2), ger);fit(ger)=pop(indexes(1),
    max_bar+1);MassGer(ger)=pop(indexes(1),max_bar+2);
}
    Print(pop,"pop","pop",-1);
    Print(fit,"fit","fitness",-1);
    Print(MassGer,"Mass","MassGer",-1);
}
```

```
void BinaryGA::give_pop_elite(Matrix &pop_elite, int &elite,
ColumnVector &indexes){
 /* Seleciona os melhores indivíduos para apróxima geração,
 garantindo diversidade entre eles.*/

    int k,c=0,j=2,i=2,l,ver=1;

    for (k=1; k<=max_bar+2; k++) {
        pop_elite(1,k)=pop(indexes(1),k);
    }

    k=1;
    while (i<=elite)  // Constrói a população de elite {
      while (c<k){
          c++;
          for (l=1; l<=max_bar; l++){
              // Verifica se os elementos da população de elite
              são iguais, garantindo a diversidade
              if (pop_elite(i-c,l)!=pop(indexes(j),l)) {
                 ver=0;
                 break;
              }
          }
          if (ver == 1){
               j++;
               c=0;
          }
      }
      for (l=1; l<=max_bar+2; l++){
          // Coloca o indivíduo na população de elite
          pop_elite(i,l)=pop(indexes(j),l);
      }
      i++;
      j++;
      k++;
      c=0;
    }
}
```

```
void Fitness::calc_fitness(Matrix &pop, Domain *pDomain){
/*  Função para calcular o fitness de uma certa população. Armazena o
valor do fitness na última coluna de pop que é retornado por
referência.*/
    int size_pop=pop.Nrows(), nconst=0,i=1,j;
    double valueOF=0, satisfy;
    int size_chrom=pop.Ncols()-2;

    ColumnVector chromossome(size_chrom);

    for (Constraints * pConstraint = theConstraintList.
    give_FirstObject();pConstraint; pConstraint =
    theConstraintList.give_NextObject())
    {
        nconst++;        // Recupera o tipo da restrição
    }

    for (i=1; i<=size_pop; i++)
    {
        // criar um pDomain com os elementos que vão estar
        ativos olhando o pop(i)
        for (j=1; j<=size_chrom; j++)
        {
            chromossome(j)=pop(i,j);
        }

        pDomain->update_PhysicalMesh(chromossome);

        PhysicalMesh *pPhyMesh=pDomain->get_pPhysicalMesh();

        int currentnumberofelements = pPhyMesh->
        get_thePhyElementList().get_numberofobjects();

        pObjectiveFunction->calc_objective_function(valueOF,
        pDomain);pop(i,size_chrom+1)=valueOF;
        pop(i,size_chrom+2)=valueOF;

        j=2;
        for (pConstraint = theConstraintList.give_FirstObject();
        pConstraint;pConstraint = theConstraintList.
        give_NextObject())
        {
            satisfy=pConstraint->calc_penalty(pDomain);
```

```
            if (pConstraint->get_type())
            {
                if (!satisfy)
                {
                    pop(i,size_chrom+1)=pop(i,size_chrom+1)+
                    pConstraint->get_weight();
                    break;
                }
                j++;
            }
            else
            {
                pop(i,size_chrom+1)=pop(i,size_chrom+1)+
                satisfy*pConstraint->get_weight();
               j++;
            }
        }
    }
}
```