UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA CIVIL

Guilherme Marques Barbaresco

OTIMIZAÇÃO DE PROBLEMAS DE ENGENHARIA UTILIZANDO O ALGORITMO DA COMPETIÇÃO IMPERIALISTA (ICA)

Florianópolis
2014

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA CIVIL

Guilherme Marques Barbaresco

OTIMIZAÇÃO DE PROBLEMAS DE ENGENHARIA UTILIZANDO O ALGORITMO DA COMPETIÇÃO IMPERIALISTA (ICA)

Trabalho de conclusão de curso submetido ao Curso de
Graduação da Universidade Federal de Santa Catarina para
A obtenção de Grau de Engenheiro Civil
Orientador: Prof. Rafael Holdorf Lopez, Dr.

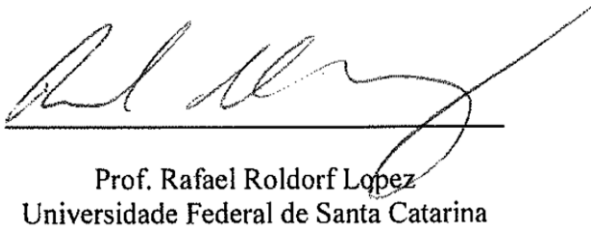
Florianópolis
2014

GUILHERME MARQUES BARBARESCO

OTIMIZAÇÃO DE PROBLEMAS DE ENGENHARIA UTILIZANDO O ALGORITMO DA COMPETIÇÃO IMPERIALISTA (ICA)

Este trabalho foi julgado adequado para a obtenção do diploma de graduação em Engenharia Civil junto à Universidade Federal de Santa Catarina.

BANCA EXAMINADORA



Prof. Rafael Roldorf Lopez
Universidade Federal de Santa Catarina
Orientador

Prof.º Leandro Fleck Fadel Miguel, Dr.
Universidade Federal de Santa Catarina

Prof.º Otávio Augusto A. da Silveira, Dr.
Universidade Federal de Santa Catarina

Florianópolis

2014

AGRADECIMENTOS

A meus amigos e familiares que me apoiaram durante o curso e me compreenderam nas ausências implicadas pela dedicação à esta graduação.

Aos colegas de graduação pela mútua parceria e apoio.

Ao meu professor e orientador Rafael H. Lopez pela grande disposição em transmitir a sua experiência e por ter acreditado no meu potencial me convidando para integrar o seu grupo de pesquisa.

Ao professor Leandro F. F. Miguel, também pela sua grande disposição em transmitir a sua experiência na área deste trabalho de conclusão de curso.

RESUMO

A crescente competitividade, consciência ambiental e escassez de recursos em diversas áreas de atuação das engenharias induziram o desenvolvimento de rigorosos métodos de tomada de decisão, como a otimização, para projetar e produzir de forma econômica e eficiente. Técnicas de otimização, por terem alcançado um elevado grau de desenvolvimento nas ultimas décadas passaram a ser utilizadas em uma ampla variedade de aplicações na indústria, incluindo aeroespacial, automotiva, química, elétrica, construção civil. Até mesmo em outras áreas, como por investidores que fazem uso desses métodos para criar portfólios que evitem o risco excessivo visando uma alta taxa de retorno. Com o rápido crescimento da tecnologia e acesso a maiores recursos computacionais, a precisão, o tamanho e complexidade dos problemas sendo resolvidos usando técnicas de otimização também aumentaram. Este trabalho apresenta conceitos sobre diferentes métodos de otimização suas vantagens e limitações, apresenta os algoritmos metaheurísticos evolutivos como o método mais adaptado atualmente para lidar com problemas de otimização global de larga-escala que representam justamente a maior parte das aplicações práticas de engenharia. E faz uso do *Algoritmo de Otimização Imperialista (ICA)* para otimizar problemas e comparar o resultado obtido com outros algoritmos de otimização com ênfase em estruturas treliçadas.

PALAVRAS-CHAVE: Otimização, imperialist competitive algorithm, metaheurístico.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A aplicação da otimização na concepção do projeto de estruturas emergiu como um proeminente campo de estudo acadêmico nas últimas décadas. Enquanto na prática ela é normalmente associada a modernas técnicas computacionais, pode-se observar a sua presença em quase todos os empreendimentos e avanços da engenharia através da história.

A necessidade de métodos formais de otimização surgem com o fato de que muitas atuações da engenharia requerem, em uma primeira instância, a solução de um problema físico, contudo essa solução poucas vezes terá uma solução única, portanto a tomada de decisão torna-se inevitável, implicando na avaliação de soluções distintas baseadas em critérios (ou objetivos). Durante a maior parte do tempo na historia da engenharia, tentativa e erro, experiência e raciocínio dedutivo auxiliaram o projetista nesse processo de tomada de decisão. (SCHOOFS, 1993)

Segundo (COELLO, 2000), o italiano Galileu Galilei (1564 – 1642) aparentemente foi o primeiro cientista que estudou a otimização de estruturas, em seu trabalho sobre a flexão de vigas Galileo Galilei (1665). Na sequencia, em um contexto bastante distante da prática de engenharia, o reconhecimento formal do conceito matemático de otimização surgiu a partir do trabalho de Leibniz, Newton, Cauchy entre outros no último quarto do milênio passado Apesar dos métodos desenvolvidos por esses pioneiros abrirem grandes possibilidades para projetistas de estruturas e engenheiros, eles não foram completamente adotados na época. Uma razão para isso foi certamente a falta de recursos computacionais para resolver os complexos problemas de otimização postos pela engenharia estrutural.

Contudo, de acordo com (RICHARDSON, 2013) outra barreira era a maneira como a engenharia civil se desenvolveu como disciplina, sempre baseada em tentativa e erro para alcançar novos avanços muitas vezes com consequências desastrosas. Além disso, existe a percepção entre os engenheiros civis que a otimização implicaria em uma *fragilidade* na estrutura*,* um equívoco oriundo de uma cultura de fatores de segurança e soluções conservadoramente aproximadas.

No passado a otimização estrutural era principalmente focada em algoritmos baseados em gradiente que são eficientes para problemas matemáticos bem definidos como a minimização de volume de uma treliça submetida somente a restrições de tensão. Nos últimos anos, a disponibilidade de novos recursos computacionais, a introdução de algoritmos não baseados em busca por gradiente e a maior conscientização do impacto que o projeto de estruturas pode ter no meio ambiente, são fatores importantes para o crescente aumento no interesse da implementação de métodos de otimização na engenharia civil.

2 OBJETIVOS

2.1 Objetivo Geral

Aplicar o Algoritmo de Competição Imperialista (ICA) na solução de problemas *benchmark* de engenharia com o objetivo de aferir o seu desempenho.

2.2 Objetivos Específicos

Este trabalho tem como objetivos específicos: apresentar o conceito de otimização, justificar a sua utilização, explicar de forma sucinta diferentes métodos de otimização, demonstrar diferentes aplicações para o uso dos métodos de otimização com ênfase na utilização de algoritmos evolutivos na otimização de treliças. Apresentar o algoritmo de otimização utilizado neste trabalho, o ICA. E através da solução de problemas matemáticos e de engenharia, fazer um comparativo de desempenho entre método adotado e outros algoritmos de otimização.

3 OTIMIZAÇÃO

De acordo com (RAO, 1996) otimizar é o ato de alcançar a melhor solução possível para um problema. Quando o contexto trata-se de problemas de engenharia os seus protagonistas, os engenheiros, devem inevitavelmente tomar decisões técnicas e logísticas durante o exercício de sua atividade. Uma vez que a maioria dos problemas enfrentados na atividade de um engenheiro pode ser descrita em função das variáveis envolvidas, a otimização pode ser definida como o processo de definir quais valores as variáveis envolvidas em um determinado problema devem possuir para que a solução encontrada seja a melhor solução possível naquilo que propõe solucionar, ou seja, a solução ótima.

Assim como convenciona a literatura acadêmica neste campo de estudo, este trabalho trata de problemas de otimização como problemas de minimização de forma a uniformizar a notação utilizada. Porém isso não limita a aplicação dos métodos de otimização para se maximizar uma função objetivo $f(x)$. Para tanto, mantêm-se o método da maneira como é aplicado para problemas de minimização da função objetivo e otimiza-se a função transformada $F(x) = -f(x)$ conforme ilustrado na Figura 1. Portanto, no contexto deste trabalho os termos "mínimo" e "ótimo" são utilizados como sinônimos.

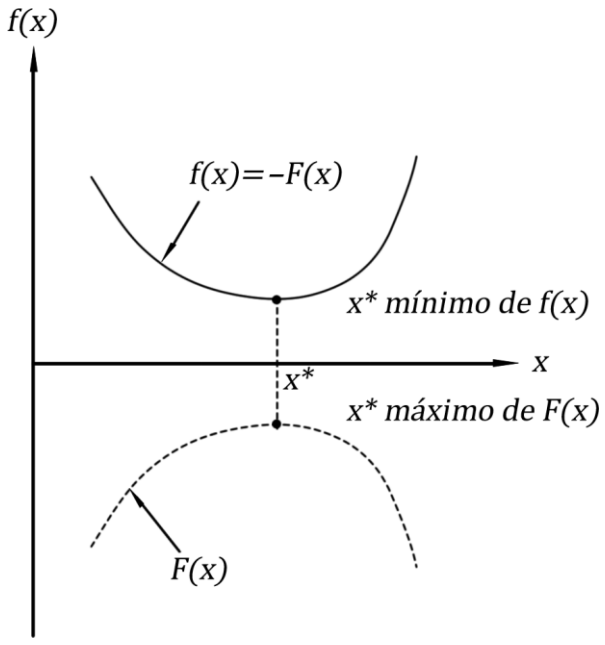


Figura 1 – Equivalência entre problemas de minimização e maximização.

3.1 A formulação do problema otimização

Segundo (ARORA, 2005) a formulação de um problema de otimização é a construção de um modelo matemático bem definido partindo das descrições físicas do problema. Essa formulação foi descrita pelo autor em cinco etapas distintas. São elas:

I. A definição do problema

A formulação do problema de otimização inicia-se pela descrição e definição do problema, que definem os requisitos que devem ser obedecidos e objetivos do projeto que devem ser alcançados.

II. Coleta de dados e informações

Informações como as tensões admissíveis, deslocamentos admissíveis, restrições arquitetônicas entre outras. São cruciais na construção do modelo matemático do projeto.

III. Identificação e definição das variáveis de projeto

O terceiro passo é a identificação e definição de todas as variáveis, chamadas *variáveis de projeto.*

Diferentes valores atribuídos as variáveis de projeto resultam em diferentes projetos. A fim de melhorar o processo de otimização as variáveis de projeto devem ser matematicamente descritas da maneira mais independente possível. Se algumas variáveis forem dependentes entre si, então os seus valores não poderão ser especificados de forma independente e isso se torna um limite para o processo de otimização.

Frequentemente existe mais de uma maneira de definir um modelo matemático para as variáveis de um projeto e este pode ser um fator relevante no desempenho do processo de otimização, tanto na qualidade do resultado da otimização quanto na eficiência computacional da sua implementação.

Equívocos na identificação e definição das variáveis de projeto podem resultar na exclusão de possíveis resultados ótimos do domínio da função sendo otimizada ou na inclusão errônea de projetos que não satisfazem todas as restrições de projeto.

IV. Identificação do critério sendo otimizado

Podem existir diversos projetos que atendam os requisitos propostos, naturalmente uns melhores que outros. E para compará-los precisamos de um critério. Na disciplina de otimização o critério precisa ser uma função escalar cujo valor pode ser obtido uma vez definidas as variáveis de projeto, esta função é a *função objetivo*, que ao

decorrer do processo de otimização será maximizada ou minimizada dependendo dos requisitos de projeto.

V. Identificação das restrições

A quinta e última etapa na formulação de um problema de otimização é a identificação das restrições do problema e o desenvolvimento das suas respectivas expressões matemáticas. Todo problema de engenharia lida com restrições nos recursos utilizados e requisitos de desempenho. Por exemplo, um elemento de concreto armado deve suportar o carregamento para qual ele foi projetado respeitando os Estados Limites Últimos, Estados Limites de Utilização, restrições arquitetônicas, entre outras restrições.

Todas as restrições devem ser descritas em função das variáveis de projeto. Alguns conceitos importantes sobre as restrições de problemas de otimização são definidos a seguir:

Linearidade das restrições: quando as restrições podem ser descritas pelas variáveis de projeto apenas como termos de primeira-ordem. Estas são chamadas *restrições lineares.* Porém, em geral problemas práticos de engenharia frequentemente têm restrições descritas pelas variáveis de projeto de maneira não linear.

Restrições de igualdade ou desigualdade: problemas de otimização podem ter restrições descritas por equações de igualdade ou de desigualdade, ou seja, por inequações. A interpretação adequada da restrição é importante para determinar se ela deve ser descrita como uma igualdade ou uma desigualdade. Contudo, em geral problemas de engenharia têm restrições de desigualdade, também chamadas de *restrições unilaterais.*

3.2 Admissibilidade do projeto

Uma vez definida a função objetivo, uma possível solução deste projeto pode ser definida simplesmente atribuindo-se valores às variáveis de projeto. Mesmo que os valores atribuídos resultem em uma solução absurda ou pouco eficiente naquilo que propõe atender, o resultado continua sendo uma solução para o projeto. Dependendo das relações entre a solução encontrada e as restrições do problema, denominamos a solução como *Admissível* se ela não violar nenhuma das restrições de projeto e *Não-Admissível* se ela o fizer.

Por exemplo, se uma solução de dimensionamento de uma viga submetida a um carregamento apresentar um deslocamento maior que o admissível, essa solução é considerada *Não-Admissível* por violar a restrição de deslocamento.

Uma solução também pode violar uma restrição apenas por atribuir um valor fora dos limites aceitáveis de uma variável, por exemplo, uma viga pode ser dimensionada com uma altura que exceda limites arquitetônicos para instalação de uma

esquadria ou algum outro elemento construtivo, esta também é uma solução *Não-Admissível.*

A constante análise da admissibilidade ou não das soluções de um projeto durante o processo de otimização é importante para evitar que o processo resulte em uma solução *Não-Admissível* e, portanto inútil em termos práticos.

3.3 Modelo geral para problemas de otimização

Com o objetivo de descrever os problemas de otimização fazemos uso de um enunciado matemático de maneira a formalizar a sua apresentação e implementação. Esse modelo matemático é definido como a minimização da função objetivo enquanto todas as restrições, sejam elas de igualdade ou desigualdade, sejam atendidas. Da seguinte maneira:

Encontre um vetor $\boldsymbol{x} \in R^n$

$$\boldsymbol{x} = (x_1, x_2, \dots, x_n) \tag{3.1}$$

contendo as *variáveis de projeto* que minimizem a *função objetivo*

$$f(\boldsymbol{x}) = f(x_1, x_2, \dots, x_n) \tag{3.2}$$

sujeitas às *p* restrições de igualdade

$$h_j(\boldsymbol{x}) = h_j\,(x_1, x_2, \dots, x_n) = 0; \quad \text{j = 1 a } p \tag{3.3}$$

às *m* restrições de desigualdade

$$g_i(\boldsymbol{x}) = h_i\,(x_1, x_2, \dots, x_n) \leq 0; \quad \text{i = 1 a } m \tag{3.4}$$

e às restrições de intervalo admissível impostas as variáveis

$$x_{ii} \leq x_i \leq x_{is} \tag{3.5}$$

Onde x_{ii} e x_{is} correspondem aos intervalos inferior e superior respectivamente da variável x_i.

3.4 Ponto Ótimo

Podemos entender um ponto *ótimo local* como sendo um ponto do domínio da função objetivo onde qualquer variação infinitesimal das *variáveis de projeto* nos leva a um resultado com desempenho igual ou inferior ao anterior.

De maneira formalizada denominamos como um ponto ótimo qualquer, um vetor $\boldsymbol{x}^*$ com n elementos:

$$\boldsymbol{x}^* = (x_1^*, x_2^*, \dots, x_n^*) \tag{3.6}$$

contendo todas as variáveis de projeto, que represente uma solução admissível, onde pode ser definida uma vizinhança V pertencente ao domínio D da função objetivo

$$V = \{x \,|x \in D \ \wedge |x^* - x| \approx 0\}. \tag{3.7}$$

onde $f(x) \geq f(x^*)$. Note que a possibilidade de igualdade entre $f(x)$ e $f(x^*)$ deve-se à possibilidade da função objetivo ter seu valor constante em uma região do seu domínio ao invés de apenas um ponto ótimo, que talvez devesse ser designada como *região ótima*, porém essa não é uma terminologia encontrada na literatura consultada neste trabalho.

3.4.1 Ótimos locais *versus* ótimos globais

Uma vez definido o problema, com suas respectivas variáveis e restrições, na grande maioria dos problemas práticos precisamos lidar com a existência de mais de um ponto ótimo local da função objetivo, conforme ilustrado na Figura 2. Podemos visualmente entender que a função possui dois mínimos locais A e C, neste caso é fácil afirmar que C é o ponto mínimo global.

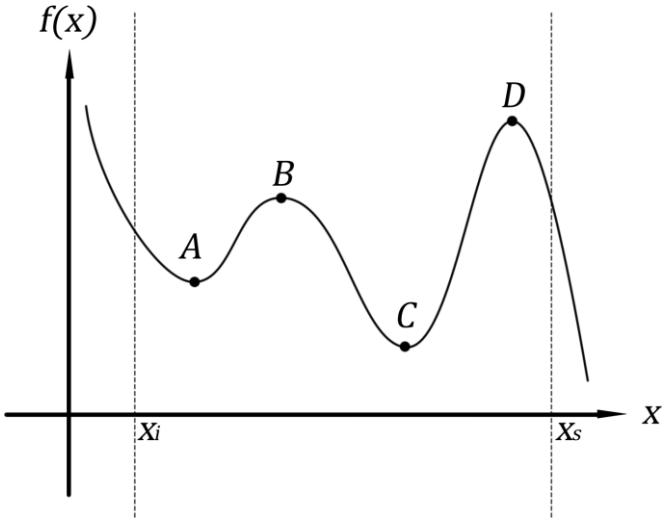


Figura 2 – Representação gráfica de pontos ótimos em A e C, onde visualmente podemos entender o ponto C como ótimo global no domínio admissível.

3.5 Representação gráfica de um problema de otimização.

Na figura 3 a seguir, está representado um problema de otimização da função de duas variáveis $f(x, y)$ sujeita à:

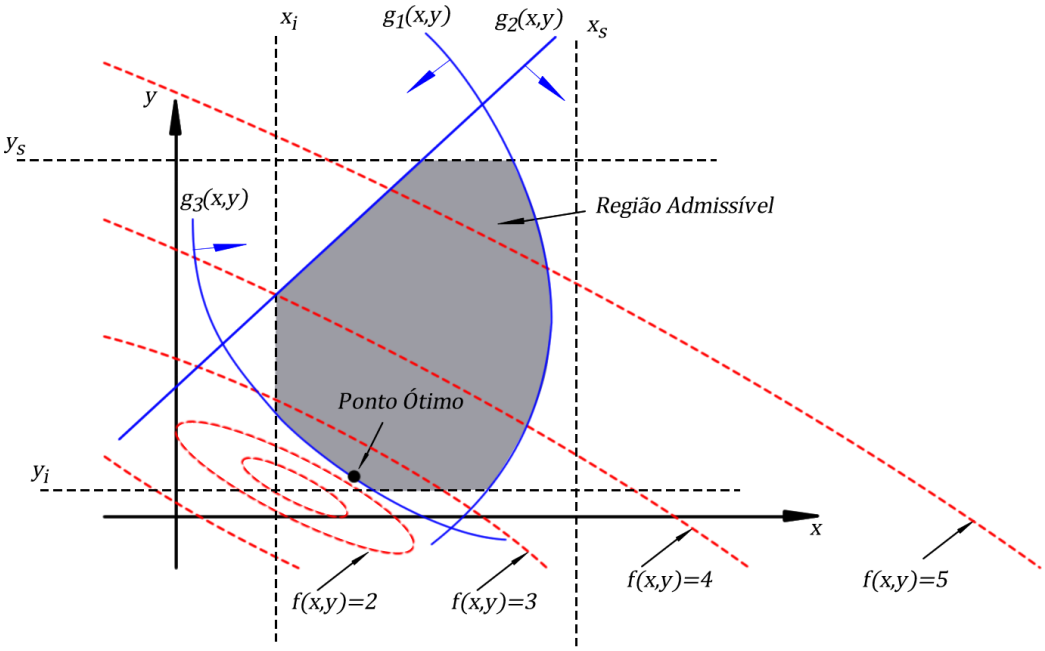
Três restrições de desigualdade:

$$g_i(x, y) < 0; \quad \text{i} = 1 \text{ a } 3$$

E aos limites inferiores e superiores impostos as variáveis:

$$x_i \leq x \leq x_s$$

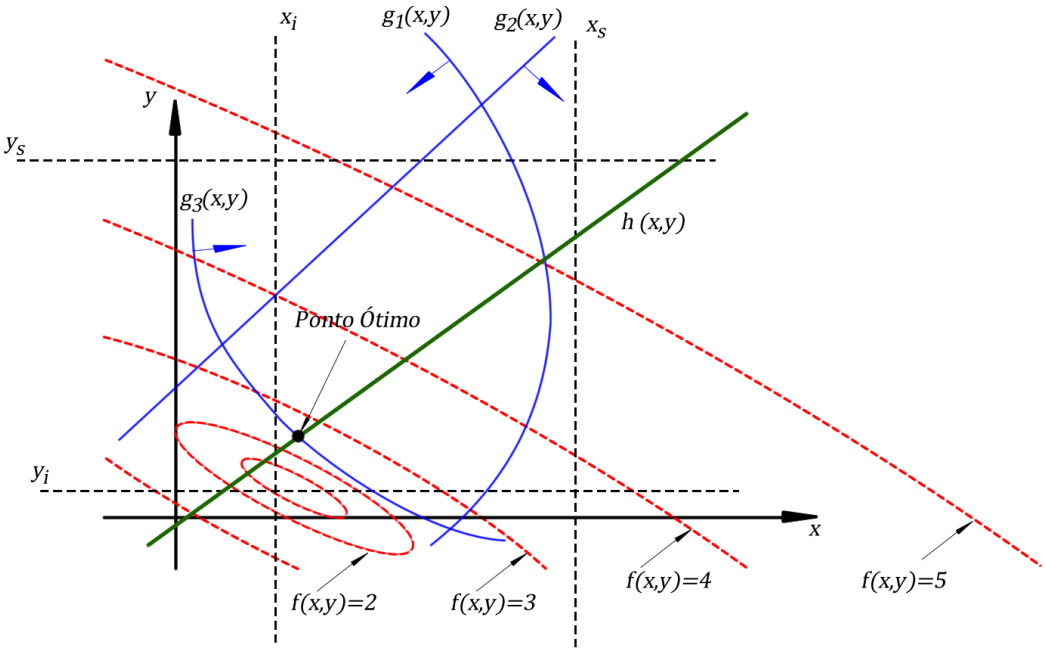
$$y_i \leq y \leq y_s$$



Curvas de nível de $f(x,y)$ representadas no plano (x,y)

Figura 3 – Representação de um problema submetido apenas a restrições de desigualdade.

Na figura 4 a seguir, está representado o problemas anterior com a adição da restrição de igualdade $h(x,y)=0$. Assim a *Região Admissível* deixa de ser representada por uma área e passa a ser representada somente pela linha definida por $h(x,y)=0$:



Curvas de nível de $f(x,y)$

Figura 4 – Representação de um problema de otimização sujeito a uma restrição de igualdade.

4 CLASSIFICAÇÕES DE UM PROBLEMA DE OTIMIZAÇÃO

Um problema de otimização pode ser classificado quanto à sua convexidade. Se o problema for definido como convexo então passa a ser denominado *problema de programação convexa*, que por sua natureza, possui apenas um ponto ótimo global.

Podemos definir uma função como convexa da seguinte maneira:

Dados dois pontos P_1 *e* P_2 pertencente ao conjunto S de soluções de $f(x, y)$, então todo o segmento de reta que conecta P_1 *e* P_2 também pertence à S.

A figura 5 mostra exemplos de conjuntos convexos e não convexos.

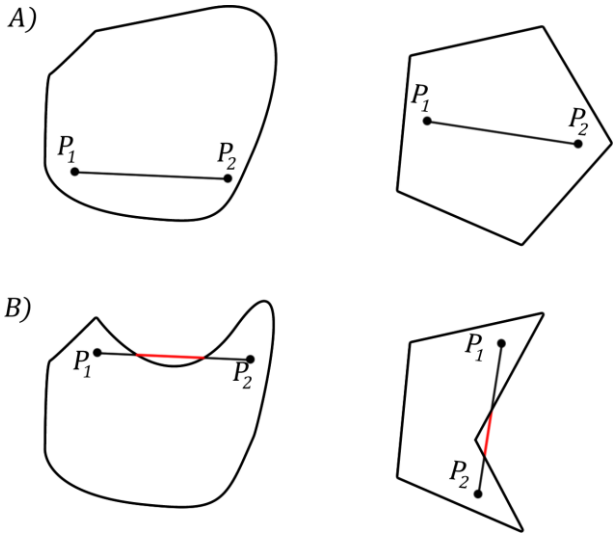


Figura 5 A) conjuntos convexos. B) conjuntos não convexos.

Seguindo a formulação de (ARORA, 2004) podemos definir a convexidade de uma função da seguinte maneira:

Teorema: Seja $f(\boldsymbol{x})$ uma função qualquer de n variáveis, onde $\boldsymbol{x}$ é um $n - vetor$. $f(\boldsymbol{x})$ é convexa se e somente se a matriz Hessiana de $f(\boldsymbol{x})$ for *semidefinida positiva* ou *definida positiva* para todo $\boldsymbol{x} \in \mathrm{R}^n$.

Entretanto, para problemas de otimização não basta apenas verificarmos a convexidade da função objetivo, já que não necessariamente é a mesma convexidade do problema de otimização. Mesmo uma função não convexa, pode ser convexa na região admissível, caracterizando um problema de otimização convexo com apenas um mínimo global. Portanto, a convexidade de um problema de otimização é observada através de uma analise da função objetivo e sua relação com as restrições do problema, da seguinte maneira:

Teorema: Seja S um conjunto definido com as restrições do modelo geral de problema de otimização na seção 3.3 onde:

$$S = \{x | h_i(\boldsymbol{x}) \ = 0, i = 1 \text{ a } p; \qquad g_j(\boldsymbol{x}) \ 0, j = 1 \text{ a } m\} \tag{4.1}$$

Então S é convexo se as funções $g_j(x)$ forem convexas e $h_i(x)$ forem lineares.

Recorrentemente aplicações práticas de engenharia recaem em problemas de otimização não convexos, ou seja, temos mais de um ótimo local. Chamamos o processo de otimização que lida com a tarefa de encontrar o ponto ótimo global entre todos os ótimos locais como *otimização global*.

Outra classificação importante de um problema de otimização é quanto à natureza das suas variáveis. Caso qualquer valor real dentro do domínio admissível possa ser atribuído à uma variável, então dizemos que esta é uma *variável contínua.* Porém existem casos onde devido a alguma imposição do problema, uma ou mais variáveis podem assumir apenas valores discretos. O exemplo clássico é o caráter discreto da área de um perfil metálico, onde, em geral existem apenas alguns perfis de área predefinida disponíveis comercialmente. Portanto a área da seção transversal desta barra metálica é considerada uma variável discreta em um problema de otimização.

A existência de variáveis discretas tem um impacto muito relevante no processo de otimização, por dificultar a aplicação direta de métodos baseados em gradiente.

Uma maneira de lidar com a presença de variáveis discretas é considerando a variável discreta como sendo contínua. Em seguida prossegue-se com o processo de otimização e então ao final os valores encontrados para as variáveis contínuas são aproximados dos valores mais próximos disponíveis e então se refaz a avaliação da admissibilidade da solução. Esta ultima etapa pode acabar gerando resultados de otimalidade inferior e recorrentemente não encontra nenhuma solução admissível.

Outra maneira de lidar com a presença de variáveis discretas é através de métodos numéricos não baseados em gradiente explicado no capítulo 5 a seguir.

5 MÉTODOS NUMÉRICOS DE OTIMIZAÇÃO

Métodos numéricos são geralmente uma sequência de operações realizadas sobre um problema matemático definido que aproxima-se de um resultado de acordo com o número de iterações, ou seja, um resultado exato só resultaria de um número infinito de iterações. Tais métodos se fazem necessários quando lidamos com problemas mais complexos como problemas de otimização da prática de engenharia, que são em geral altamente não lineares. Métodos iterativos podem se basear no gradiente da função analisada, chamamos estes métodos de *Métodos baseados em gradiente*, que são métodos que devem ser empregados caso a função tenha apenas um ótimo global. Ou podem ter seu funcionamento independente da avaliação do gradiente da função objetivo, são os *Métodos não baseados em gradiente.* O método empregado neste trabalho enquadra-se nesta última categoria.

5.1 Métodos não baseados em gradiente

Segundo (Conn et al. 2009), a grande maioria dos problemas práticos de engenharia são problemas de otimização não convexos. Ainda assim, mesmo sob tais circunstancias existe o interesse de prosseguir com a otimização. E para fazê-lo precisamos utilizar técnicas não baseadas em gradiente.

De acordo com (Conn et al. (2009)

> "É quase uma perversidade natural que os problemas práticos de hoje são complexos, não-lineares, não-convexos e não suficientemente explícitos para fornecerem derivadas confiáveis. De fato, tais problemas sempre foram numerosos, mas, 30 anos atrás, quando as técnicas de otimização eram relativamente mais ingênuas do que hoje, mesmo os mais otimistas praticantes não tentariam otimizar tais problemas complexos. Mas não em 2008! A diversidade de aplicações incluem problemas de engenharia, matemática, física, química, economia, finanças, medicina, transporte, ciência da computação, negócios e pesquisa operacional."

Note que a citação está contextualizada para o ano de 2008, quando foi escrita.

Portanto, pela presença de variáveis discretas, e pela não convexidade dos problemas práticos; uma classe de métodos de otimização não-linear chamadas de *métodos de otimização não baseados em gradiente* se fazem necessários. De fato, pela sua recorrente necessidade, métodos numéricos não baseados em gradiente são considerados umas das mais importantes, abertas e desafiadoras áreas da engenharia e ciência da computação, e com enorme potencial prático.

5.2 Algoritmos Heurísticos e Metaheurísticos

De acordo com (Conn et al. 2009), algoritmos Heurísticos são uma classe de algoritmos de otimização não baseados em gradiente que se tornaram o "último recurso" para lidar com problemas complexos como a otimização global de larga-escala, que são problemas de otimização com funções objetivo altamente não lineares, geralmente envolvendo variáveis discretas, cuja avaliação da função objetivo possui um elevado custo computacional.

Algoritmos heurísticos são estratégias iterativas de otimização global que, partindo de um chute inicial, fazem uma varredura do domínio da função objetivo seguindo uma 'heurística', ou seja, um conjunto de regras combinando informações extraídas das iterações anteriores de modo a fazer com que as iterações seguintes tendam a procurar nas regiões mais "promissoras" do domínio da função a fim de fazer uso do custo computacional da maneira mais eficiente possível.

A denominação "Metaheurístico" primeiramente proposto por (GLOVER, 1986) é resultado da composição de duas palavras em Grego. *Heurístico* que deriva do verbo *heuriskein* (ευρισκειν) que significa "encontrar", enquanto o sufixo *meta* significa

"além" ou "em uma instância superior". Antes de o termo ser amplamente adotado, metaheurístico era normalmente referida como *heurística moderna*.

(GLOVER, 1986) define que um algoritmo metaheuristico se diferencia do heurístico por ter sua estratégia baseada em um fenômeno natural, seja ele químico, físico ou biológico.

Entre os métodos metaheurísticos existem aqueles que fazem uso de analogias evolutivas, ou seja, a população de soluções sendo analisadas a cada iteração adapta-se e sofre as consequências de sua adaptação de maneira análoga a *sobrevivência do mais adaptado* como na teoria da evolução Darwiniana. Segundo (BLUM e ROLI, 2003) estes são denominados algoritmos *Metaheurísticos Evolutivos*, esta é a classificação à qual pertence o algoritmo utilizado neste trabalho.

Métodos metaheurísticos também podem ser distinguidos entre aqueles que são *determinísticos*, ou seja, para uma dada população inicial e função objetivo, sempre teremos a mesma solução final; e *estocásticos*, que são algoritmos que envolvem o uso de parâmetros aleatórios durante o processo de otimização. De acordo com esta classificação o algoritmo utilizado neste trabalho enquadra-se como *estocástico*.

Segundo (BLUM e ROLI, 2003) em relação ao desempenho de otimização global, existem duas características desejáveis em todo algoritmo metaheurístico, são as capacidades de *diversificação* e *intensificação.* A capacidade de *diversificação* se refere à capacidade de gerar boas soluções e também à de explorar o máximo possível do espaço de busca do problema garantindo que o ótimo global seja encontrado. A capacidade de *intensificação* se refere à capacidade de, uma vez encontradas regiões de mínimos locais, definir de fato a melhor solução em cada região destas.

Os avanços desses algoritmos incluem os seguintes:

(i) Eles não requerem uma análise de gradiente da função otimizada, portanto podem ser aplicados à problemas onde o gradiente é difícil de ser obtido ou simplesmente não pode ser definido;
(ii) Eles não ficam "presos" em ótimos locais se corretamente calibrados;
(iii) Eles podem ser aplicados a funções descontínuas e não lineares;
(iv) Ao final do processo de otimização eles podem fornecer um conjunto de soluções ótimas (ótimos locais), dando ao projetista mais opções a serem escolhidas.

Porém os métodos metaheurísticos também têm as suas limitações:

(i) Eles requerem um ajuste de diversos parâmetros utilizados para regular o seu funcionamento, e esses parâmetros variam de problema para problema e, portanto, acabam sendo definidos por tentativa e erro.
(ii) Uma estimativa *a priori* do seu desempenho ainda é um problema em aberto.

(iii) Um número muito grande de avaliações da função objetivo pode ser necessário para otimizar problemas mais complexos. O que pode tornar o método inviável dependendo do custo computacional de cada avaliação da função objetivo.

6 ALGORITMO DA COMPETIÇÃO IMPERIALISTA (ICA)

O Algoritmo da Competição Imperialista é um algoritmo metaheurístico, estocástico, evolutivo, proposto por Esmaeil Atashpaz Gargari professor e pesquisador da Universidade do Teerã. Assim como a maioria dos algoritmos evolutivos o ICA é particularmente eficiente para a solução de problemas de otimização global de larga-escala.

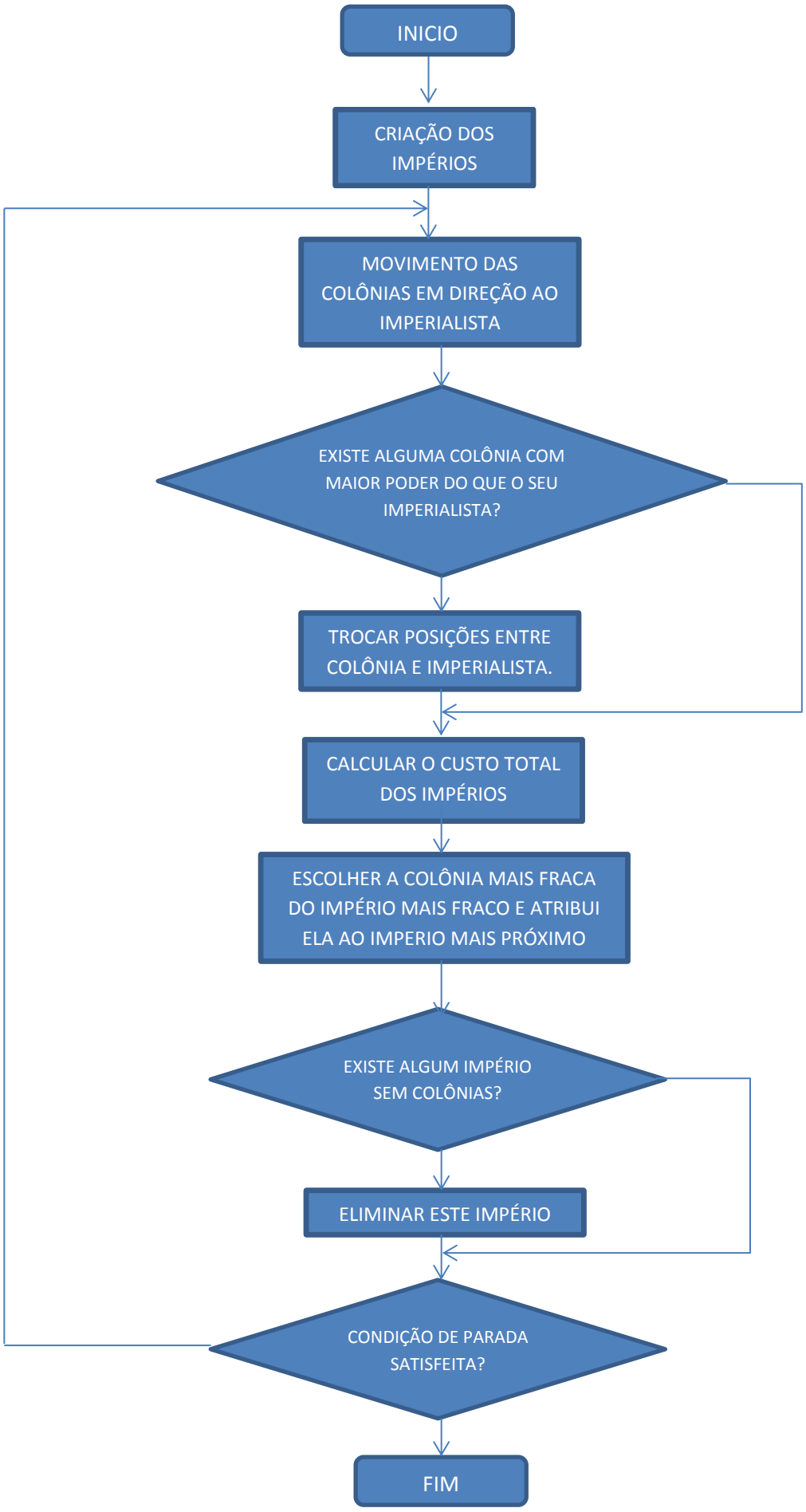
A Heurística do ICA faz analogia a conceitos de relações sociopolíticas e pode ser visto como a contraparte sociopolítica do Algoritmo Genético que, como a maioria dos algoritmos evolutivos, tem sua heurística baseada em fenômenos biológicos.

O método tem início com a criação de uma população inicial. No ICA cada solução sendo avaliado durante o processo de otimização é referido como *País*. A partir da primeira população, tomam-se os países com os melhores valores e estes passam a ser denominados *Imperialistas* e todos os demais países são designados *Colônias.* O valor da função objetivo em cada Imperialista define o *Poder* deste Imperialista. Em seguida todas as colônias são atribuídas aos imperialistas de maneira proporcional ao Poder de cada Imperialista, fazendo com que o Imperialista com maior Poder fique com o maior número de Colônias e o conjunto (Imperialista + Colônias) passa a ser designado *Império*.

Após dividir todas as Colônias entre os Imperialistas, as Colônias movimentam-se a cada iteração, aqui denominada *década,* em direção ao seu respectivo Imperialista. O Poder total de cada imperialista passará a ser uma combinação do Poder do Imperialista com o poder de suas Colônias.

E então através de uma comparação de poderes entre os Impérios, de maneira concomitante ao movimento das colônias, a competição imperialista retira as colônias dos impérios mais fracos e as anexa ao Império com maior Poder. Resultando na convergência de todos os países em um só grande império com um único Imperialista onde, espera-se que esteja o ponto ótimo da função objetivo.

A seguir, o fluxograma do ICA sintetiza o funcionamento do algoritmo:


INICIO
CRIAÇÃO DOS IMPÉRIOS
MOVIMENTO DAS COLÔNIAS EM DIREÇÃO AO IMPERIALISTA
EXISTE ALGUMA COLÔNIA COM MAIOR PODER DO QUE O SEU IMPERIALISTA?
TROCAR POSIÇÕES ENTRE COLÔNIA E IMPERIALISTA.
CALCULAR O CUSTO TOTAL DOS IMPÉRIOS
ESCOLHER A COLÔNIA MAIS FRACA DO IMPÉRIO MAIS FRACO E ATRIBUI ELA AO IMPERIO MAIS PRÓXIMO
EXISTE ALGUM IMPÉRIO SEM COLÔNIAS?
ELIMINAR ESTE IMPÉRIO
CONDIÇÃO DE PARADA SATISFEITA?
FIM

6.1 Parâmetros do algoritmo

Todo algoritmo metaheurístico têm seu funcionamento regido por um conjunto de parâmetros. A seguir são apresentados os parâmetros do ICA:

Numero de países: Parâmetro que define a quantidade de soluções analisadas a cada iteração, o número de países é diretamente proporcional ao custo computacional.

Número inicial de imperialistas: Parâmetro que define quantos dos países iniciais são definidos como *Imperialistas*. Um número maior neste parâmetro resulta em uma maior a capacidade de *diversificação* ao custo de uma necessidade de mais iterações para que ocorra a convergência do algoritmo.

Décadas: Define a quantidade de iterações do processo de otimização.

Taxa de Revolução: Define a taxa com a qual as características sociopolíticas de um país muda.

Coeficiente de assimilação: Define o "tamanho do passo" dado por cada colônia a cada década.

Coeficiente angular de assimilação: Define o limite máximo do ângulo aleatório existente entre o movimento da colônia e a direção do imperialista a cada iteração.

Zeta: Define a proporção dada ao somatório do poder de todas as colônias de um império em relação ao poder do *Imperialista n*a definição do poder dos Impérios.

6.2 Movimentação das colônias

A cada iteração todas as colônias se movimentam em direção aos seus respectivos imperialistas. Esse movimento é regido por dois parâmetros do algoritmo: o Coeficiente de assimilação(β) e o Coeficiente angular de Assimilação(γ).

O "tamanho do passo" dado por cada colônia é um valor aleatório escolhido entre 0 e $(\beta \times d)$ onde d é o vetor distância entre o imperialista e a colônia, e β é um valor maior que 1. Um valor de β maior que um faz com que as colônias se aproximem do imperialista por ambos os lados.

A direção do movimento feito por cada colônia é direção da colônia até o imperialista adicionado de um desvio aleatório (θ) escolhido entre $(-\gamma, \gamma)$.

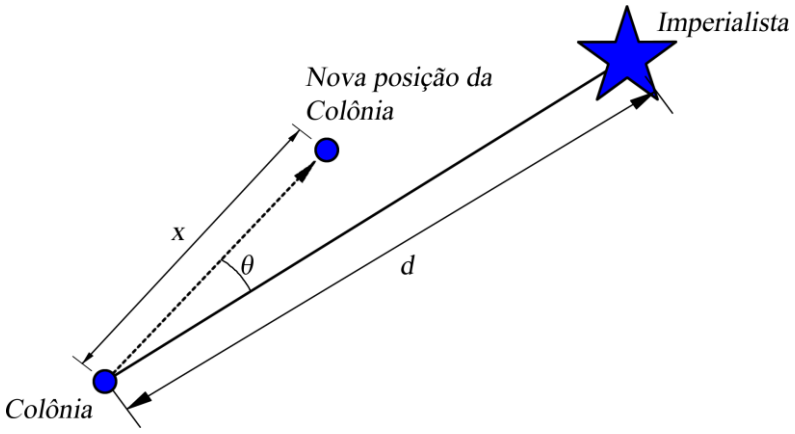


Figura 6 - Ilustração do deslocamento de uma colônia.

7 MÉTODO DA PENALIZAÇÃO

Como o ICA da maneira como ele é proposto pelo autor não é adaptado para lidar diretamente com problemas com restrições, neste trabalho fez-se o uso de uma técnica para lidar com restrições chamado *método da penalização.* De maneira sucinta o método procede da seguinte maneira:

Dada uma função objetivo $f(\boldsymbol{x})$ formulada conforme convencionado na seção 3.3:

No método da penalização substitui-se a função objetivo $f(\boldsymbol{x})$ pela função transformada:

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + P$$

Onde P é a penalização da solução sendo avaliada. Caso a solução tenha violado alguma restrição, terá um valor positivo, caso contrário será nulo. P é formalmente descrito da seguinte maneira:

$$: \sum_{j=1}^{p} \left(a_j \times \left|h_j(\boldsymbol{x})\right|\right) + \left(\begin{cases} 0 & , g_i(\boldsymbol{x}) \leq 0 \\ \sum_{i=1}^{m} \left(b_i \times g_i(\boldsymbol{x})\right), & g_i(\boldsymbol{x}) > 0 \end{cases} \right) \tag{7.1}$$

Onde a e b são coeficientes de penalização definidos para cada restrição. Que em geral são definidos empiricamente, ou seja, arbitram-se coeficientes e verifica-se se são suficientes para evitar que o algoritmo resulte em soluções *não admissíveis*. Ou arbitram-se valores altos o suficiente para tornar qualquer solução que viole uma restrição pouco "promissora" para a heurística de otimização.

Através de um exemplo elaborado pelo autor deste trabalho a figura 7 a seguir ilustra a visualização da transformação de $f(x) = x^2$ em $F(x) = f(x) + P$ através da inclusão do coeficiente de penalização b para a função submetida à restrição de desigualdade $g_i(x) = 50 - x < 0$:

Note como o aumento do coeficiente de penalização aproxima o mínimo da função não restringida ao mínimo da função restringida até que o coeficiente b é suficiente para igualar os dois valores.

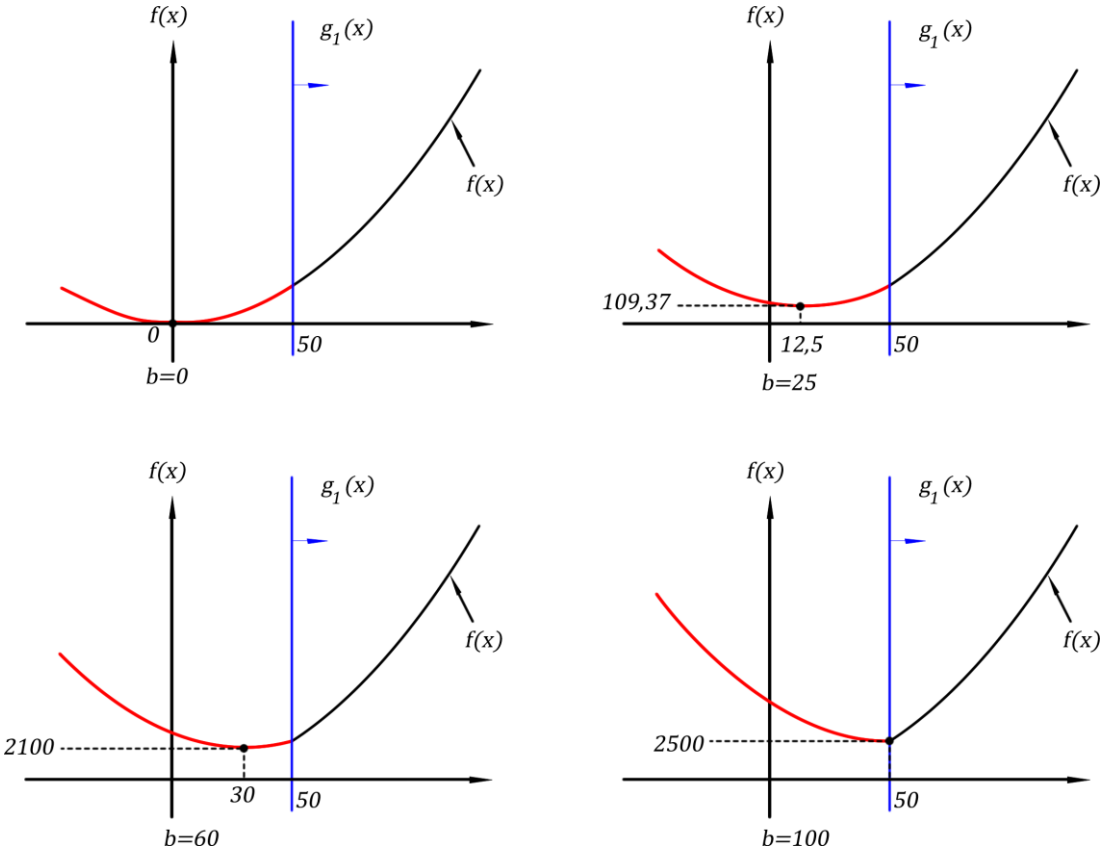


Figura 7 – Transformação da função objetivo restringida devido à aplicação de uma penalização.

8 OTIMIZAÇÃO DE ESTRUTURAS TRELIÇADAS

Uma treliça é um arranjo estrutural composto por barras, normalmente formando triângulos. Teoricamente, assume-se que as barras de uma treliça são conectadas umas as outras através de ligações rotuladas sem atrito. Na vida real, no entanto, as conexões são muito menos livres devido à solda ou ao parafusamento das conexões. Mesmo com alguma rigidez nas conexões, a análise de uma estrutura através de um modelo rotulado pode ser utilizado com precisão desde que o centro de gravidade

das barras conectadas encontrem-se sobre a posição onde é colocado o modelo de rótula considerado no cálculo. Por esta característica, desde que os esforços sejam atuantes somente nas conexões, os elementos que compõe uma treliça são submetidos apenas a esforços de tração ou compressão (HEYDEN, 2005).

Segundo (CHRISTENSEN, 2008) a otimização estrutural de uma treliça pode ser separada em três diferentes parâmetros sendo otimizados, a seguir uma explicação de cada um desses parâmetros é dada:

Otimização Dimensional: É a otimização das seções transversais das barras da treliça como variáveis de projeto.

Otimização Geométrica: É a otimização das posições dos nós como variáveis de projeto

Otimização de Topologia: É a otimização do número de nós e como estes nós são conectados uns aos outros.

As imagens a seguir ilustram os conceitos descritos acima:

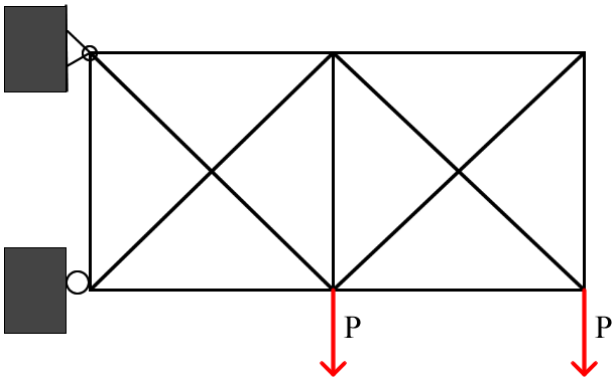


Figura 8 – Treliça original.

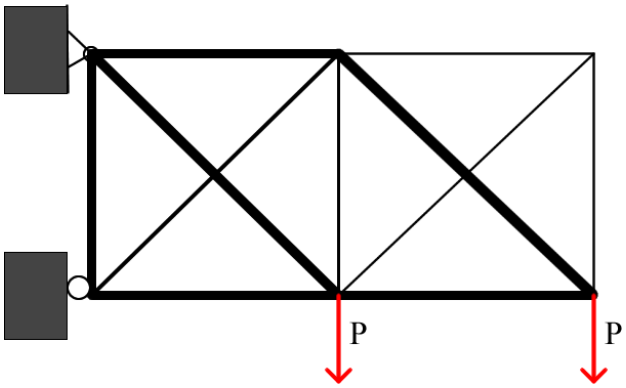


Figura 9 – Otimização dimensional, escolhendo as áreas dos perfis.

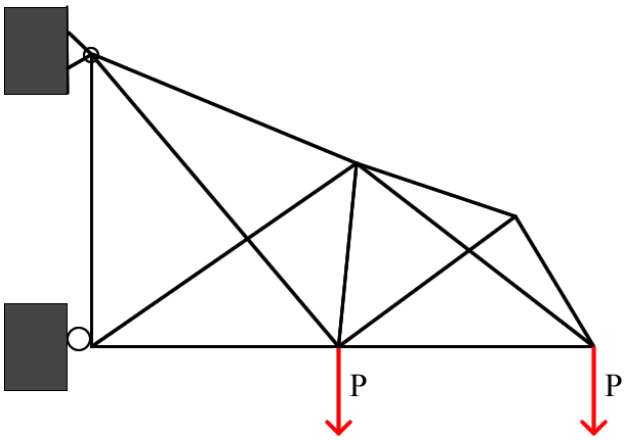


Figura 10 – Otimização geométrica, definindo as posições dos nós.

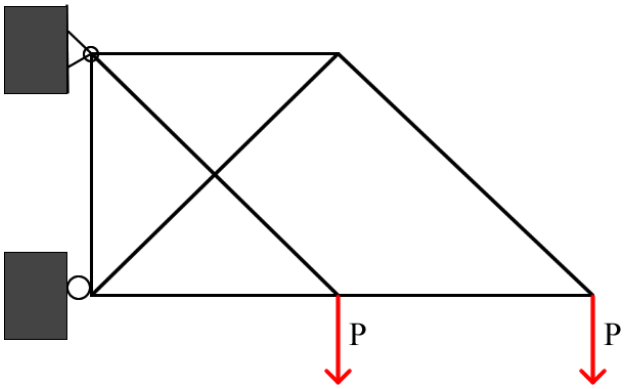


Figura 11 – Otimização de Topologia, onde a interconectividade dos nós é otimizada.

Uma maneira de otimizar os três parâmetros, é levar em consideração um parâmetro de cada vez, começando geralmente com a topologia. Esta técnica chama-se otimização em multi-níveis(também chamada de otimização em camadas (Kawamura et al. 2002). Entretanto, é evidente que esta abordagem nem sempre atinge a solução ótima, uma vez que os problemas da maneira como foram divididos não são linearmente separáveis (DEB e GULATI, 2000). Uma das vantagens dos metaheurísticos é poder lidar com a otimização de todos estes parâmetros de forma simultânea.

A otimização de topologia é geralmente realizada através do *Método da estrutura base* – trata-se da definição inicial de todos os possíveis nós e conexões entre estes nós formando uma estrutura base. Partindo desta estrutura base o processo de otimização altera as seções transversais das barras, e as posições dos nós definidas inicialmente em busca da solução ótima, podendo retirar barras e também nós que se mostrem desnecessários.

Quanto maior a quantidade e interconectividade dos nós da estrutura base, maior a quantidade de *variáveis de projeto* terá a *função objetivo* e, portanto, um maior custo computacional será necessário para que a convergência seja atingida. Entretanto,

reduzir a quantidade e interconectividade dos nós da estrutura base, potencialmente excluirá possíveis soluções ótimas do problema.

A imagem a seguir ilustra duas diferentes estruturas base para uma treliça apoiada em A e B responsável por resistir aos carregamentos P1, P2 e P3, a estrutura base 2 apresentando mais barras do que a estrutura base 1.

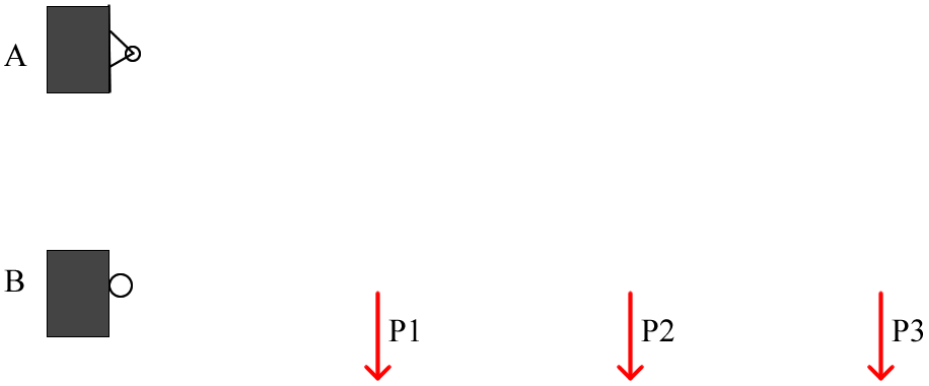


Figura 12 – Definições da treliça proposta.

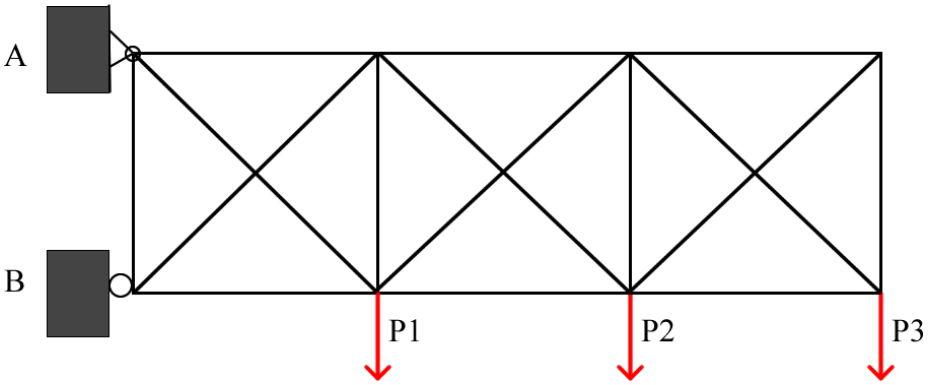


Figura 13 – Estrutura Base 1.

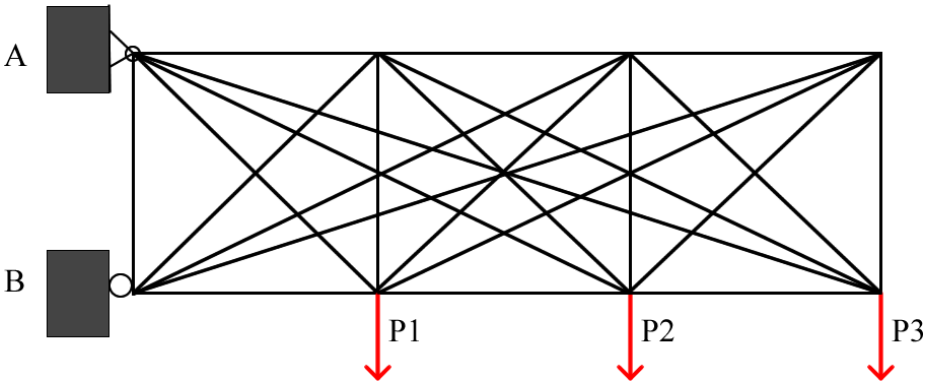


Figura 14 – Estrutura Base 2.

Podemos observar que a *Estrutura Base 2* fornecerá um maior número possíveis soluções, o que demandaria um maior numero de avaliações do algoritmo de otimização para alcançar a solução ótima, porém forneceria um maior potencial para encontrar uma solução com desempenho superior à *Estrutura Base 1*.

9 PROBLEMAS ESTUDADOS:

Com o propósito de testar a eficiência do ICA, este trabalho compara o desempenho deste algoritmo com outros algoritmos evolutivos na solução de problemas matemáticos e de engenharia, os chamados problemas *benchmark*. Que são problemas padrão utilizados como referência.

Devido ao caráter estocástico do método utilizado o seu desempenho é avaliado e comparado com outros métodos heurísticos de maneira estatística.

É importante frisar que o *número de avaliações* é um parâmetro utilizado para comparação de desempenho de algoritmos de otimização. Seu valor denota o custo computacional do processo de otimização e é calculado através do produto entre o número de soluções sendo avaliadas a cada iteração e número de iterações.

Devido à natureza estocástica do ICA (assim como de outros metaheurísticos), a convergência do algoritmo é ligeiramente diferente cada vez que o processo de otimização é performado. Sendo que *convergência* é a curva formada pelo valor da função objetivo versus número de iterações. Assim, para cada exemplo avaliado neste trabalho uma convergência típica é ilustrada e não devemos esperar que estas convergências possam ser exatamente reproduzidas mesmo utilizando a mesma rotina fornecida em anexo neste trabalho.

Todos os problemas analisados foram implementados com uso do software MATLAB e resolvidos 200 vezes, e dos resultados foram extraídos o melhor valor, valor médio e desvio padrão para fins comparativos.

9.1 Problemas Benchmark matemáticos:

Neste campo acadêmico ao decorrer do seu desenvolvimento, diversas funções matemáticas foram elaboradas com o objetivo de analisar o desempenho dos algoritmos de otimização sob condições intencionalmente adversas, como a presença de muitos ótimos locais, para que se pudesse ter uma melhor referência da real capacidade de otimização global dos algoritmos sendo desenvolvidos. Duas funções foram utilizadas neste trabalho para analisar o desempenho do ICA, a função Griewank e a função Shekel.

Os valores utilizados para comparação para os problemas 9.1.1 e 9.1.2 foram retirados de (KARABOGA e AKAY, 2009)

9.1.1 Função Griewank:

A função Griewank é um função multidimensional, multimodal, contínua, proposta por (GRIEWANK, 1981). Essa função tem sido amplamente utilizada para testar a convergência de algoritmos de otimização devido ao seu grande número de mínimos locais que crescem exponencialmente de acordo com numero de variáveis.

A função n-dimensional é definida da seguinte maneira:

$$f_n(\boldsymbol{x}) = \left(\frac{1}{4000}\right)\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{10.1}$$

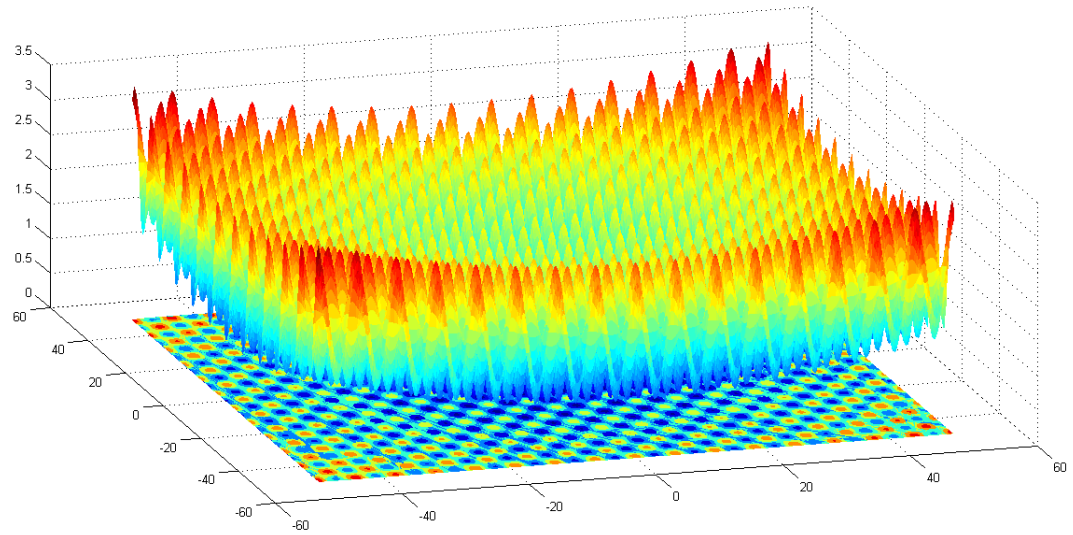


Figura 15 – Função Griewank para 2 variáveis ilustrada no intervalo (-50,50), com sua respectiva projeção no plano $(x, y) = (0,0)$.

A função normalmente considerada na literatura, e também neste trabalho, para comparação de desempenho de otimização contém 30 dimensões avaliadas no intervalo $[-600, 600]$.

O valor ótimo para esta função encontra-se em $\boldsymbol{x}^* = (0, 0, \ldots, 0)$, $onde\ f(\boldsymbol{x}^*) = 0$.

Na **tabela 1** a seguir estão os parâmetros de desempenho do ICA ao solucionar a função de Griewank, em comparação a outros algoritmos de otimização:

Tabela 1 – Média e Desvio Padrão representam o valor da função objetivo.

Algoritmo	Média	Desvio padrão	Número de avaliações
GA	10,633	1,1614	500.000
PSO	0,017	0,021	500.000
DE	0,001	0,002	500.000
ABC	0,000	0,000	500.000
ICA	**0,000**	**0,000**	**500.000**

As siglas utilizadas na **tabela 1** correspondem aos seguintes algoritmos metaheurísticos:

GA – Genetic Algorithm

PSO – Particle Swarm Optimization

DE – Differential Evolution

ABC – Artificial Bee Colony

Podemos observar que o ICA teve um desempenho tão bom quanto o ABC, encontrando o ótimo global em todos os processos de otimização.

A figura a seguir representa uma convergência típica do método na solução da função de Griewank. Em vermelho está representado o valor mínimo encontrado a cada iteração, a linha pontilhada representa a média dos valores avaliados a cada iteração.

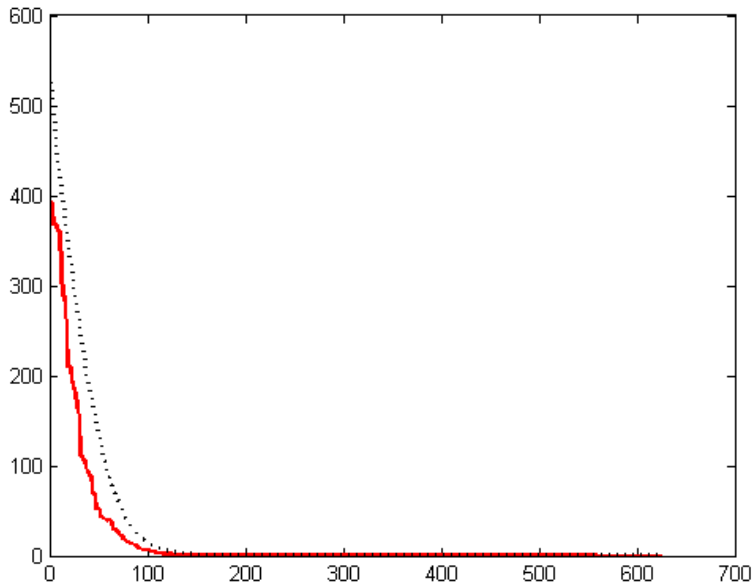


Figura 16 – Convergência típica para a solução da função Griewank alcançada pelo ICA.

Com a finalidade de ilustrar o funcionamento do algoritmo, a Figura 17 a seguir exibe o processo de otimização feita pelo ICA para a função de Griewank com $n = 2$ variáveis, $pop. = 20\ países$ e $Iterações = 400\ Décadas$. Resultando em um numero de avaliações igual a 8000.

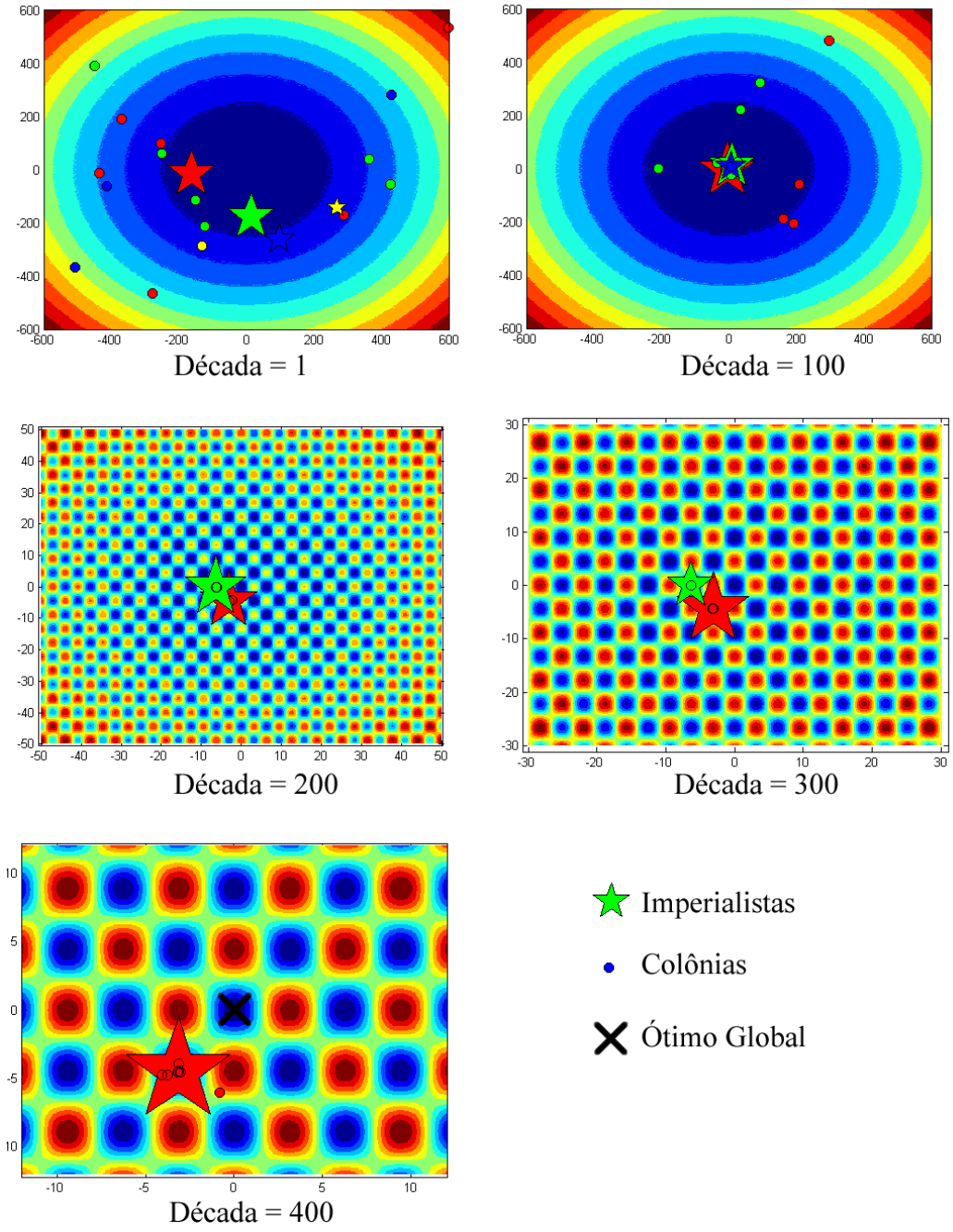


Figura 17 – Visualização do processo de otimização da função Grieawank.

Note que neste exemplo o processo de otimização não convergiu para o resultado ótimo global da função objetivo, ao invés disto encontrou um ótimo local bastante próximo do ótimo global. Pode-se deduzir deste resultado que o número de avaliações (8000) não foi suficiente para a definição do mínimo global.

9.1.2 Função Shekel:

A função Shekel é uma função multidimensional, multimodal, contínua, proposta por (SHEKEL, 1971). Também é uma função amplamente utilizada para testar a convergência de algoritmos de otimização devido a sua grande capacidade de customização uma vez que o numero de mínimos locais e dimensões da função são facilmente alterados na sua implementação.

A função n-dimensional contendo m mínimos locais é definida da seguinte maneira:

$$f(\boldsymbol{x}) = -\sum_{i=1}^{m}\left(\sum_{j=1}^{n}\left(x_j - C_{ji}\right)^2 + \beta_i\right)^{-1}, \quad onde \tag{10.2}$$

$$m = 10,$$

$$\beta = \frac{1}{10}(1,2,2,4,4,6,3,7,5,5)^T,$$

$$C = \begin{pmatrix} 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 3 & 1 & 2 & 3 \\ 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 3 & 1 & 2 & 3 \end{pmatrix}.$$

O valores de β e C utilizados são os mesmos da literatura usada no comparativo de desempenho na Tabela 2. Para $n = 5$ dimensões $f(\boldsymbol{x}) = -10{,}15$ no ponto ótimo.

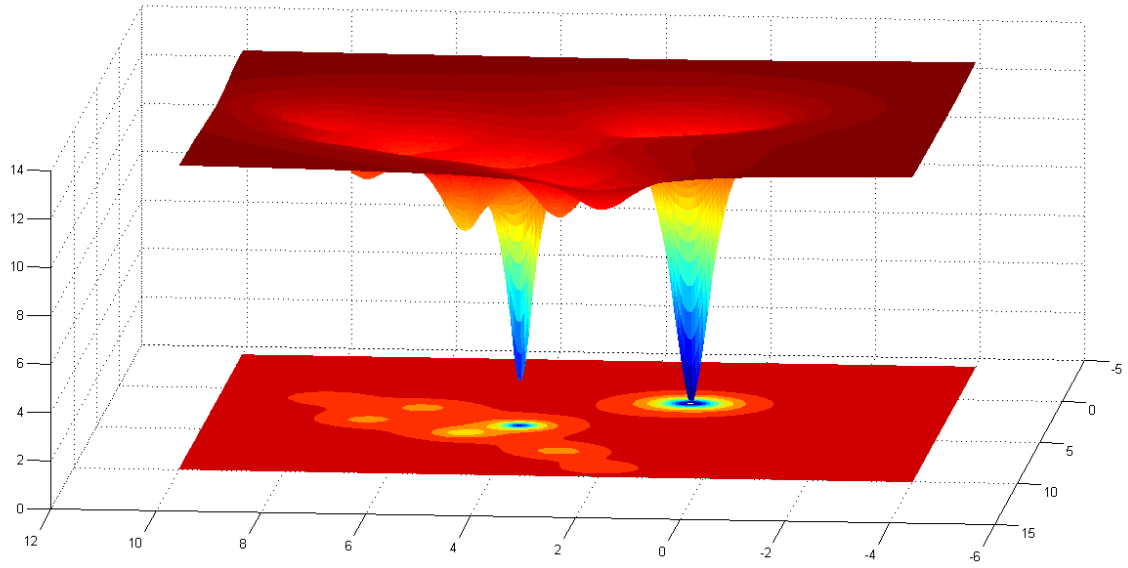


Figura 18 - Função Shekel com 2 dimensões e 10 ótimos locais.

A Tabela 2 a seguir exibe os parâmetros de desempenho do ICA ao solucionar a função Shekel fazendo a comparação com outros algoritmos de otimização.

Tabela 2 – Comparação de desempenho de otimização para a função Shekel.

Algoritmo	Média	Desvio padrão	Número de avaliações
GA	-5,66052	3,866737	500000
PSO	-2,0870079	1,178460	500000
DE	-10,1532	0	500000
ABC	-10,1532	0	500000
ICA	-10,1532	0	500000

9.2 Problemas Benchmark com restrições:

Diversos problemas de engenharia, que são em geral problemas mecânicos modelados de maneira simplificada, são utilizados como problemas *benchmark* para otimização. Fundamentalmente também se trata da otimização de um problema matemático, porém com as restrições impostas pelo modelo físico. Dois problemas destes foram utilizados neste trabalho, presentados a seguir.

Os valores foram utilizados para comparação de desempenho dos exemplos 9.2.1 e 9.2.2 foram retirados de (Sadollah et al. 2013).

9.2.1 Viga soldada:

O Problema consiste na execução de uma viga metálica de perfil retangular maciço A suportada por uma solda em um suporte rígido B. A viga deve suportar a carga P aplicada à distância L com uma deflexão máxima admissível de 0,25 polegadas no ponto de aplicação da carga. O critério sendo otimizado neste problema é o custo de execução da viga. Os custos unitários de viga e solda são padronizados, meramente didáticos e mantidos os mesmo valores da literatura para uma comparação coerente dos resultados.

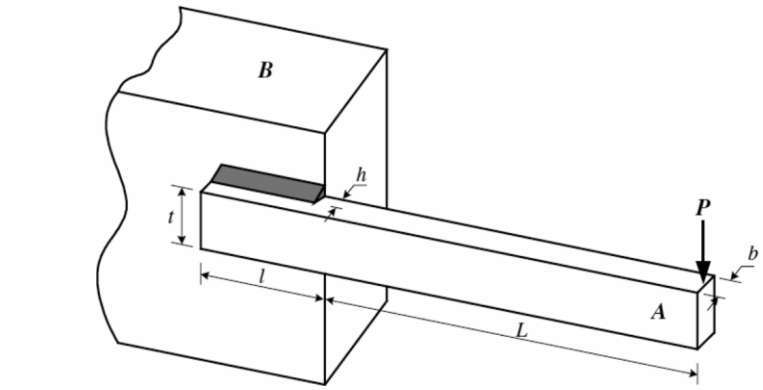


Figura 19 – Problema da viga soldada.

O problema possui quatro *variáveis de projeto:*

- A largura do perfil da viga – *b*
- O comprimento de barra no apoio – *l*
- A altura do perfil – *t*
- E a largura da perna da solda – *h*

Portanto o processo de otimização consiste em encontrar o vetor

$$\boldsymbol{x} = (x_1, x_2, x_3, x_4) = (h, l, b, t)$$

Que minimize o custo da viga definido pela equação:

$$f(\boldsymbol{x}) = (1 + C_1)\, h^2 l + C_2\, tb\, (L\ + l) \tag{10.3}$$

Sujeito às sete restrições:

1- Tensão de cisalhamento admissível:

$$g_1(\boldsymbol{x}) = \ \tau(\boldsymbol{x}) - \tau_{max} \leq 0$$

$$\text{Onde } \ \tau_{\max} = 13.600 \text{ psi}$$

2- Tensão normal admissível:

$$g_2(\boldsymbol{x}) = \ \sigma(\boldsymbol{x}) - \sigma_{max} \leq 0$$

$$\text{Onde } \ \sigma_{\max} = 30.000 \text{ psi}$$

3- Restrição construtiva:

$$g_3(\boldsymbol{x}) = h - b \leq 0$$

4- Restrição de custo da viga:

$$g_4(\boldsymbol{x}) = f(\boldsymbol{x}) - 5 \leq 0$$

5- Restrição construtiva:

$$g_5(\boldsymbol{x}) = 0{,}125 - h \leq 0$$

6- Restrição de deslocamento admissível:

$$g_6(\boldsymbol{x}) = \delta(\boldsymbol{x}) - \delta_{max} \leq 0$$

Onde $\delta_{\max} = 0{,}25$ pol.

7- Carca crítica de flambagem*:*

$$g_7(\boldsymbol{x}) = P - P_c(x) \leq 0$$

Onde $P = 6000\ lb$

Limites impostos as variáveis:

$$0{,}1 \leq x_i \leq 2, \qquad i = 1{,}4$$

$$0{,}1 \leq x_i \leq 10, \qquad i = 2{,}3$$

Constantes do problema:

- Custo unitário de execução da viga:

$$C_1 = 0{,}10471\ (\$/\text{pol}^3)$$

- Custo unitário de execução da solda:

$$C_2 = 0{,}04811\ (\$/\text{pol}^3)$$

G	12×10^6 psi
L	14 pol
E	30×10^6 psi
P	6000 lb

Onde:

$\tau(\boldsymbol{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{l}{2R} + (\tau'')^2}$	$\tau' = \frac{P}{\sqrt{2}hl}$	$\tau' = \frac{MR}{J}$
$J = 2\left\{\sqrt{2}hl\left[\frac{l^2}{12} + \left(\frac{h+t}{2}\right)^2\right]\right\}$	$R = \sqrt{\frac{l^2}{4} + \left(\frac{h+t}{2}\right)^2}$	$\delta(\boldsymbol{x}) = \frac{4PL^3}{Ebt^3}$
$P_c(\boldsymbol{x}) = \frac{4{,}013E\sqrt{\frac{t^3b^4}{36}}}{L^2} \times \left(1 - \frac{t}{2L}\sqrt{\frac{E}{4G}}\right)$	$\tau' = P(L + \frac{l}{J})$	$\sigma(\boldsymbol{x}) = \frac{6PL}{bt^3}$

Na tabela 3 a seguir estão os parâmetros de desempenho do ICA ao otimizar a função da viga soldada, em comparação à outros algoritmos de otimização.

Tabela 3 – Comparação de desempenho de otimização para a função da viga soldada.

Algoritmo	Pior Resultado	Média	Melhor Resultado	Desvio Padrão	Numero de Avaliações
GA3	1.785835	1.771973	1.748309	1.12E−02	900,000
CAEP	3.179709	1.971809	1.724852	4.43E−01	50,020
CPSO	1.782143	1.748831	1.728024	1.29E−02	240,000
aHPSO	1.814295	1.749040	1.724852	4.01E−02	81,000
PSO-DE	1.724852	1.724852	1.724852	6.7E−16	66,600
NM-PSO	1.733393	1.726373	1.724717	3.50E−03	80,000
MGA	1.9950	1.9190	1.8245	5.37E−02	N.A
SC	6.3996785	3.0025883	2.3854347	9.6E−01	33,095
DE	1.824105	1.768158	1.733461	2.21E−02	204,800
UPSO	N.A	2.83721	1.92199	0.683	100,000
CDE	N.A	1.76815	1.73346	N.A	240,000
ES	N.A	1.777692	1.724852	8.8E−02	30,000
ABC	N.A	1.741913	1.724852	3.1E−02	30,000
TLBO	N.A	1.72844676	1.724852	N.A	10,000
MBA	1.724853	1.724853	1.724853	6.94E−19	47,340
ICA	1.728622	1.725753	1.724853	1,29E-3	50.000

A seguir, a figura 20 ilustra uma convergência típica do problema da viga soldada em função do número de décadas. Em vermelho está representado o valor mínimo encontrado a cada iteração, a linha pontilhada representa a média dos valores avaliados a cada iteração.

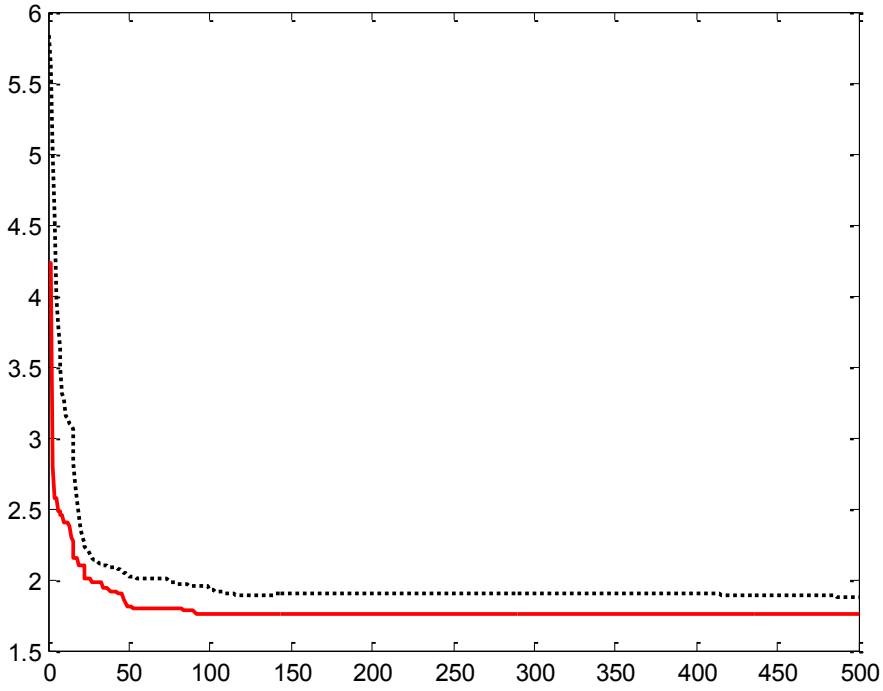
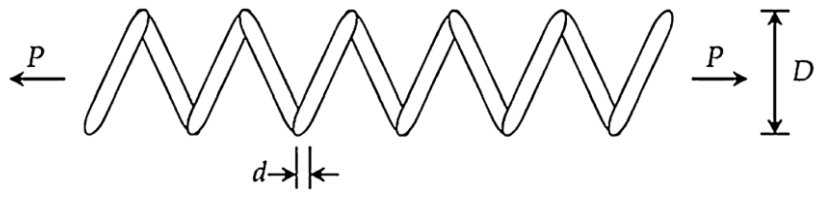


Figura 20 – Convergência típica do problema da viga soldada otimizada pelo ICA.

9.2.2 Mola tração/compressão

Este problema consiste no dimensionamento de uma mola submetida à um carregamento axial P. O propósito deste problema é minimizar a massa da mola, sendo que o projeto está sujeito a restrições de deflexão mínima, cisalhamento admissível, frequência de vibração, limites no diâmetro externo da mola, e construtivas. As variáveis de projeto são: O diâmetro do fio (d), o diâmetro da mola (D) e o número de voltas da mola (N).

Figura 21 – Mola tração/compressão.



A formulação deste problema se dá da seguinte maneira:

Minimizar:

$$f(x) = (N + 2)Dd^2 \tag{10.4}$$

Sujeito às restrições:

$$g_1(x) = 1 - \left(\frac{D^3 N}{71785 d^4}\right) \leq 0$$

$$g_2(x) = \left(\frac{4D^2 - dD}{12.5666(D^2 d^3 - d^4)}\right) + \left(\frac{1}{5108 d}\right) - 1 \leq 0$$

$$g_3(x) = 1 - \left(\frac{140{,}45 d}{DN}\right) \leq 0$$

$$g_4(x) = \left(\frac{d + D}{1{,}5}\right) - 1 \leq 0$$

$0{,}05 \leq d \leq 2{,}00$ polegadas.

$0{,}25 \leq D \leq 1{,}30$ polegadas.

$2{,}00 \leq N \leq 15{,}00$

Resultados:

Na Tabela 4 a seguir estão os parâmetros de desempenho do ICA ao solucionar a função *Mola tração/compressão*, em comparação à outros algoritmos de otimização.

Tabela 4 – Comparação de desempenho de otimização para a função *Mola tração/compressão*.

Algoritmo	Pior avaliação	Melhor avaliação	Média	Desvio padrão	Número de avaliações
GA	0.0128220	0.0127048	0.0127690	3.94E−05	900,000
CAEP	0.0151160	0.0127210	0.0135681	8.42E−04	50,020
CPSO	0.0129240	0.0126747	0.0127300	5.20E−04	240,000
HPSO	0.0127190	0.0126652	0.0127072	1.58E−05	81,000
NM-PSO	0.0126330	0.0126302	0.0126314	8.47E−07	80,000
QPSO	0.018127	0.012669	0.013854	0.001341	2000
PSO	0.071802	0.012857	0.019555	0.011662	2000
DE	0.012790	0.0126702	0.012703	2.7E−05	204,800
SC	0.016717272	0.012669249	0.012922669	5.9E−04	25,167
ABC	N.D	0.012665	0.012709	0.012813	30,000
TLBO	N.D	0.012665	0.01266576	N.D	10,000
MBA	0.012900	0.012665	0.012713	6.30E−05	7650
ICA	0.0130946	0.012665	0.012713	3.61E−05	7600

As siglas utilizadas na Tabela 4 correspondem aos seguintes algoritmos metaheurísticos:

Tabela 5 – Legenda dos algoritmos utilizados.

GA	Genetic Algorithm
CAEP	Cultural Algorithm Evolutionary Programing
CPSO	Combinatorial particle swarm optimization
HPSO	Hierarchy Particle Swarm Optimization
NM-PSO	Nelder-Mead Particle Swarm Optimization
QPSO	QPSO stands for Quantum Particle Swarm Optimization
PSO	Particle Swarm Optimization
DE	Differential Evolution
SC	Supply chain optimization
ABC	Artificial Bee Colony
TLBO	Teaching-learning-based optimization
MBA	Mine Blast Algorithm
ICA	Imperialist Competitive Algorithm

A figura 22 a seguir mostra uma convergência típica deste problema em função do número de décadas obtida pelo ICA. Em vermelho está representado o valor mínimo encontrado a cada iteração, a linha pontilhada representa a média dos valores avaliados a cada iteração.

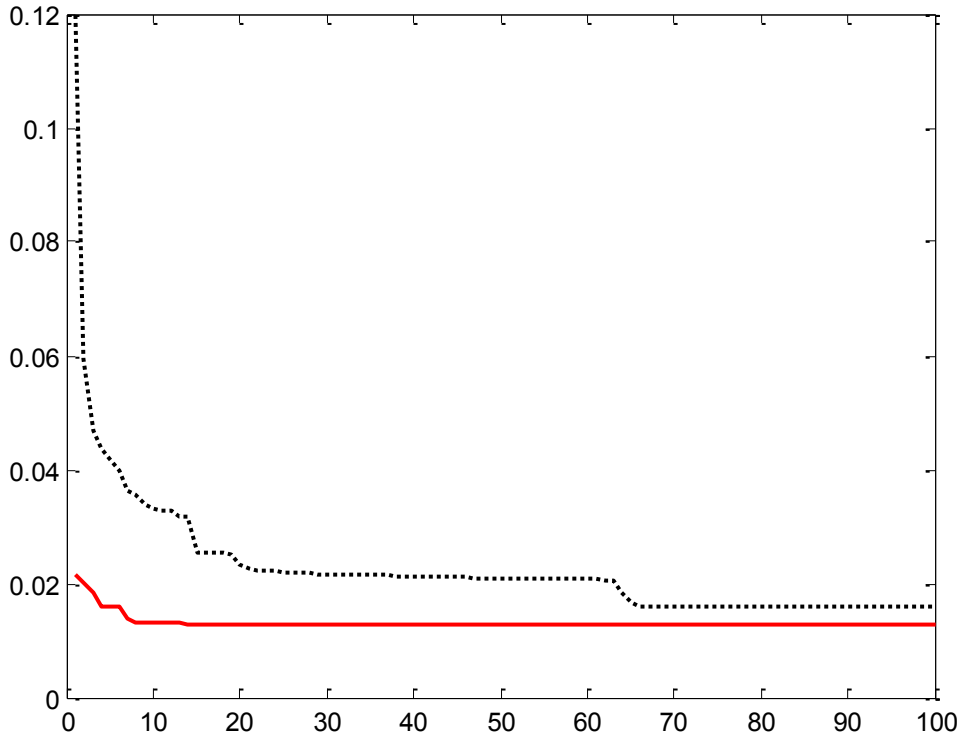


Figura 22 – Convergência típica do *Mola tração/compressão* otimizado pelo ICA.

9.3 Estruturas treliçadas

Os exemplos 9.3.1 e 9.3.2 são comparados com resultados retirados de (MIGUEL et al. 2013). As treliças foram calculadas através do uso do método dos deslocamentos e o peso próprio da estrutura foi desconsiderado do cálculo.

9.3.1 *Treliça de 10 barras* – Dimensionamento, Geometria e Topologia

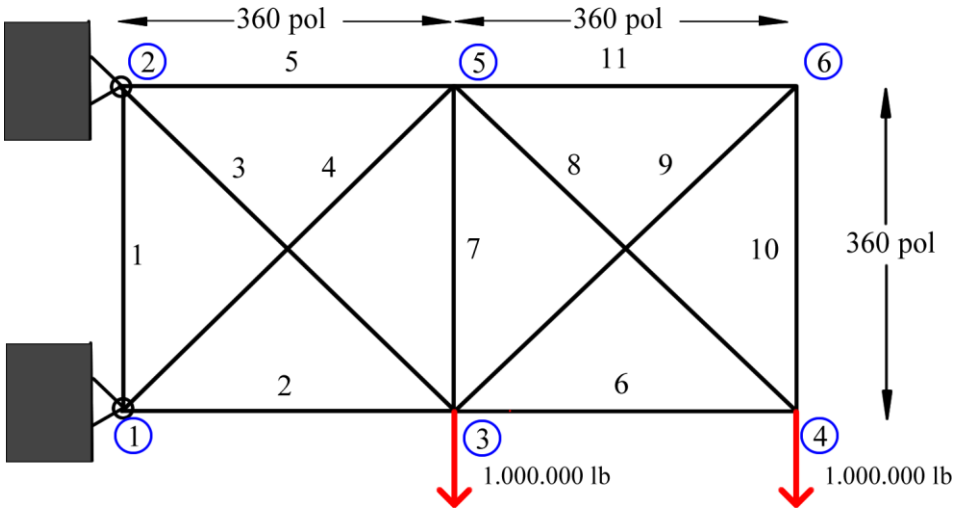


Figura 23 – Estrutura base para o problema 10.3.1.

Esta treliça tem sido frequentemente utilizada como problema *benchmark* na avaliação de desempenho de algoritmos de otimização e aparece na maioria das publicações acadêmicas sobre otimização de estruturas. A estrutura base utilizada para este exemplo é mostrada na Figura 23 e os parâmetros de projeto são representados na tabela 6. Neste exemplo é realizada a otimização de *dimensionamento, geometria e topologia* de forma simultânea. A treliça deve suportar os dois carregamentos impostos de 1.000.000 lb nos nós 3 e 4 respeitando as tensões admissíveis em cada perfil e o deslocamento máximo admissível de 2 polegadas nos nós de aplicação das cargas, estas portanto são restrições deste problema de otimização.

Assim como na bibliografia consultada e cujos resultados são comparados na tabela 7, neste trabalho manteve-se os critérios de otimização de geometria utilizando as coordenadas verticais dos três nós superiores (2, 5 e 6) da treliça como variáveis de projeto, permite-se o deslocamento vertical destes nós entre 180 e 1000 polegadas da origem arbitrada no nó 1. A Figura 24 a seguir ilustra a região admissível para os nós superiores.

Tabela 6 – Parâmetros do problema 10.3.1

Parâmetros de projeto	Valor
Módulo de elasticidade	10^4 ksi
Densidade	0,1 lb/pol^3
Tensão normal admissível	25 ksi
Deslocamento vertical admissível nos nós 3 e 4	2,0 pol

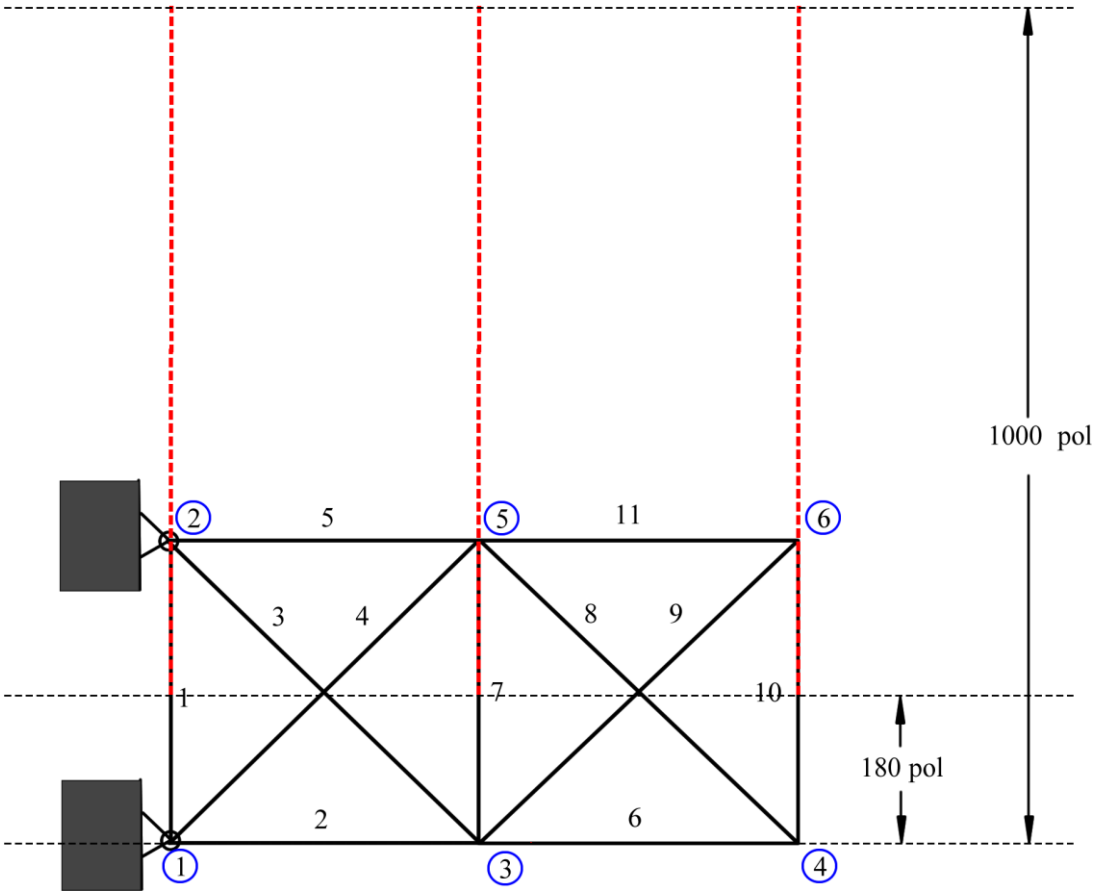


Figura 24 – A região admissível para a variação de posição dos nós superiores da treliça encontram-se na região destacada em vermelho.

As seções transversais das barras disponíveis são valores inteiros escolhidos entre 0 pol² e 30 pol² para serem utilizadas na otimização *Dimensional*. Note que as seções das transversais das barras portanto são variáveis discretas.

Diferente de algumas das fontes bibliográficas consultadas para este problema, neste trabalho foi executada um processo de otimização unimodal, ou seja apenas o resultado ótimo foi extraído do processo.

A tabela a seguir compara o desempenho do ICA com o de outros algoritmos.

Tabela 7 – Comparação de desempenho de otimização para o problema 10.3.1

	Melhor	Média	Coef. var.	Número de avaliações
FA	2705,000	2893,457	2,12%	50.000
ICA	2726,927	2861,478	2,44%	50.000
HS	2893.45	N.D	N.D	4.075
GA1	3254	N.D	N.D	3.840
GA2	2736	N.D	N.D	500.000

N.D – Valor não disponível.

Onde as sigla *FA* refere-se ao *FireFly Algorithm* empregado por (MIGUEL et al. 2013).

O resultado ótimo encontrado pelo ICA ao fim do processo de otimização deste problema consiste na combinação de variáveis de projeto exibidas nas Tabelas 9 e 10, a seguir:

Tabela 8 – Posições dos nós do resultado ótimo encontrado pelo ICA para o problema 10.3.1

Nó	**Coordenadas (polegadas)**	
	x	**y**
1	0	0
2	0	787,36
3	360	0
4	720	0
5	360	478,85

Tabela 9 – Áreas das barras adotadas no resultado ótimo pelo ICA para o problema 10.3.1.

Barra	**Area (pol²)**	**Tensão atuante (ksi)**
2	12	10,51
3	3	10,34
4	6	19,15
5	12	10,47
6	8	9,43
9	14	9,26

A estrutura ótima é ilustrada na Figura 25 a seguir, note que o nó 6 e as suas respectivas barras foram completamente removidos pelo processo de otimização.

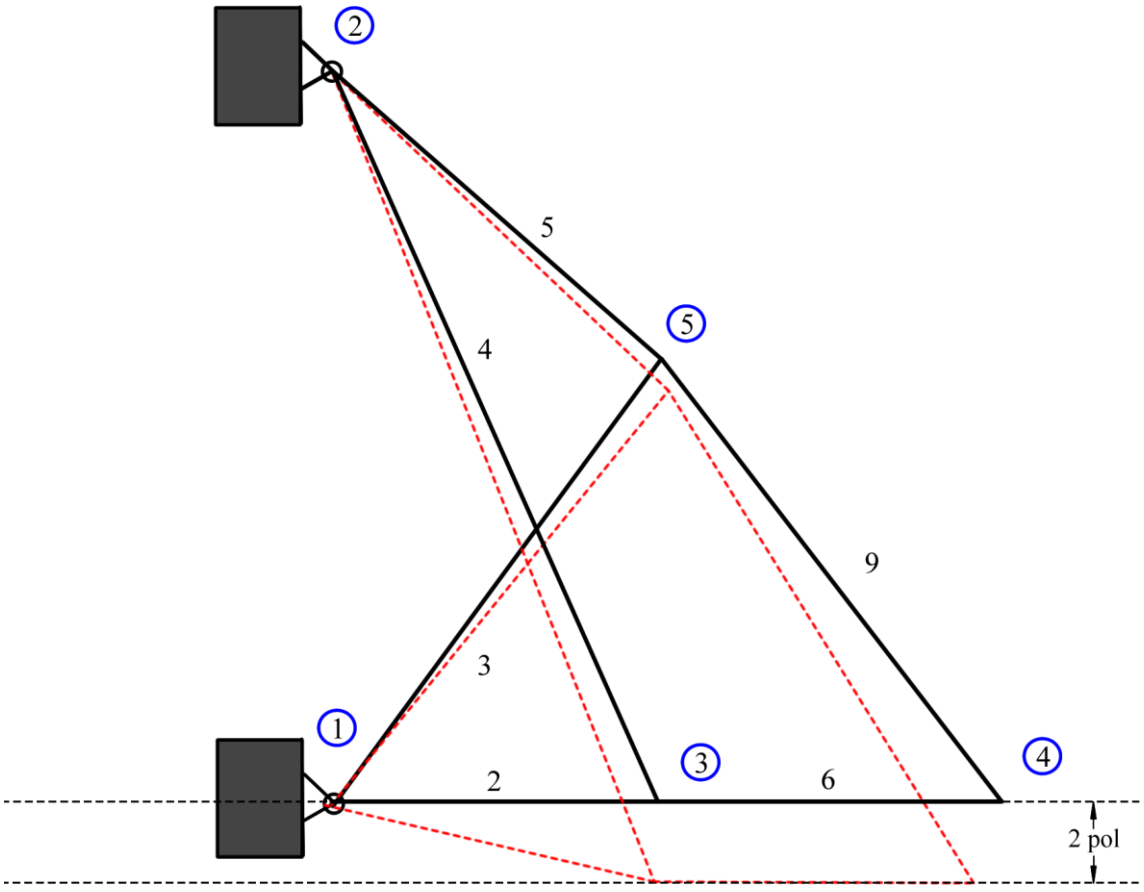


Figura 25 – Desenho da estrutura ótima encontrada pelo ICA para o problema 10.3.1. Em tracejado vermelho foi exibida a estrutura deformada após a aplicação do carregamento.

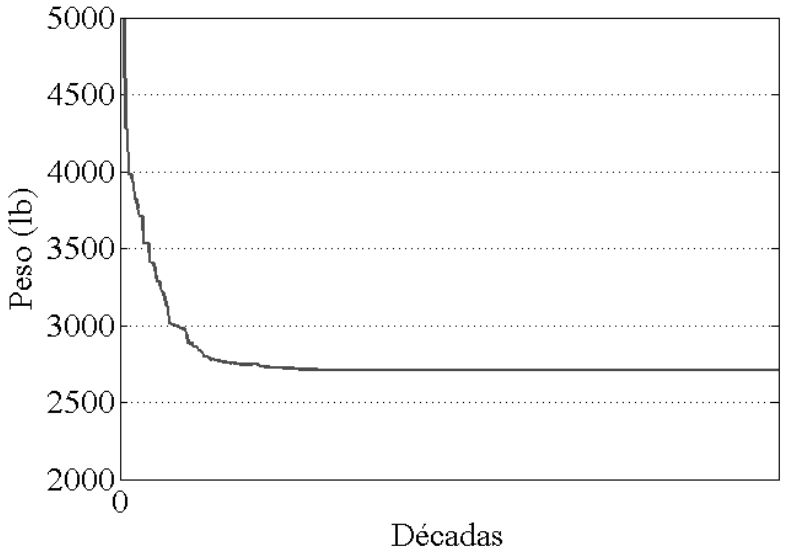


Figura 26 – Converência típica para o problema 10.3.1 otimizado pelo ICA.

9.3.2 *Treliça de 15 barras* – Dimensionamento, Geometria e Topologia

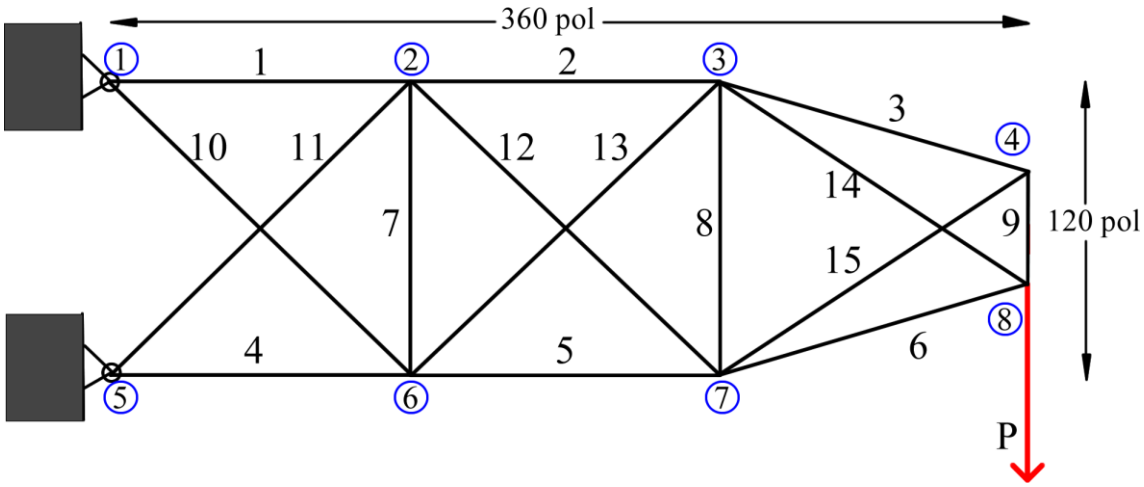


Figura 27 – Estrutura base para o problema 10.3.2.

Este problema, proposto por (Miguel, 2013), consiste em uma estrutura de 15 barras cuja *estrutura base* é ilustrada na figura 27. As variáveis de projeto são: as coordenadas x e y dos nós 2, 3, 6 e 7 e as coordenadas y dos nós 4 e 8; Os perfis das 15 barras; E a topologia das 15 barras.

As coordenadas x dos nós 6 e 7 são igualadas as dos nós 2 e 3, assim como na literatura consultada.

A treliça é apoiada em A e B, e deve suportar o carga $P = 10.000\ lb$ vertical para baixo aplicada no nó 8. Respeitando a tensão máxima admissível de 25 ksi em cada perfil, e o deslocamento máximo admissível no ponto de aplicação da carga.

Neste exemplo, além das restrições de tensão e deslocamentos admissíveis, considerou-se a restrição quanto à falha de cada elemento por instabilidade, assim como na bibliografia utilizada (Miguel, 2013) para comparação na tabela 11. Utilizou-se tensão crítica de flambagem de Euler calculada pela equação X:

$$\sigma_{Cr} = \frac{100EA_i}{8L_i^2} \tag{10.5}$$

As seções transversais disponíveis são representadas no conjunto D a seguir em polegadas:

$D = (0.111, 0.141, 0.174, 0.220, 0.270, 0.287, 0.347, 0.440, 0.539, 0.954, 1.081,$

$1.174, 1.333, 1.488, 1.764, 2.142, 2.697, 2.800, 3.131, 3.565, 3.813, 4.805, 5.952,$

$6.572, 7.192, 8.525, 9.300, 10.850, 13.330, 14.290, 17.170, 19.180)$.

As restrições de posição impostas aos nós são 100 pol ≤ x2 ≤ 140 pol, 220 pol ≤ x3 ≤ 260 pol, 100 pol ≤ y2 ≤ 140 pol, 100 pol ≤ y3 ≤ 140 pol, 50 pol ≤ y4 ≤ 90 pol, -20 pol ≤ y6 ≤ 20 pol, -20 pol ≤ y7 ≤ 20 pol, and 20 pol ≤ y8 ≤ 60 pol. Sendo que esses são os valores das distâncias de cada nó até a origem arbitrada no nó 5. Através desta definição a Figura 28 a seguir ilustra a possível região dos nós. Note que os nós 1 e 2 são fixos; os nós 2, 3, 6 e 7 devem ficar dentro da região demarcada; e os nós 4 e 8 variam apenas no eixo *y* dentro de um domínio de 40 polegadas estabelecido para cada um.

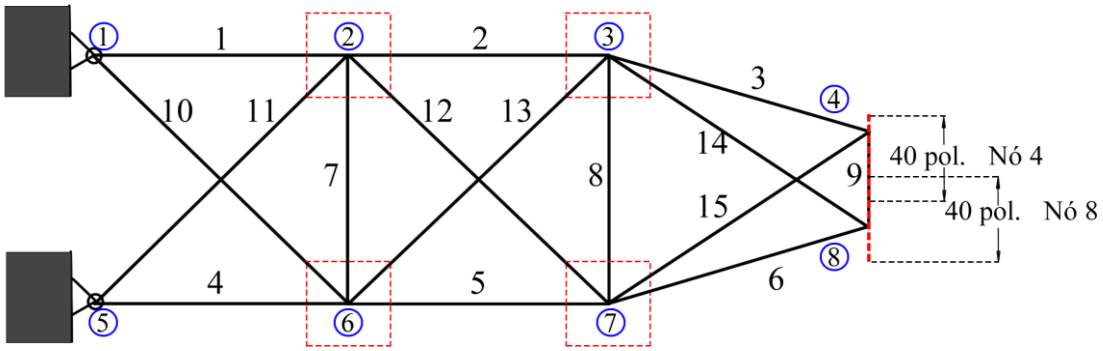


Figura 28 – Domínio de cada nó sendo otimizado representado pelo tracejado vermelho.

Tabela 10 – Comparativo de desempenho de otimização do problema 10.3.2.

	Melhor	**Média**	**Coef. Var.**	**Num. De Avaliações**
ICA	118.331	153,74	8,13%	8.000
FF	125.229	152.61	8.69%	8.000

O resultado ótimo encontrado pelo ICA ao fim do processo de otimização deste problema consiste na combinação de variáveis de projeto exibidas nas tabelas 11 e 12, a seguir:

Tabela 11 – Coordenadas dos nós do resultado ótimo encontrado pelo ICA para o problema 10.3.2.

Nó	Coordenadas (polegadas)	
	x	y
1	0	3,048
2	2,882	2,54
3	5,588	2,54
4	9,144	1,386
5	0	0
6	2,882	-0,194
7	5,588	0,508
8	9,144	1,387

Tabela 12 – Áreas das barras encontradas para o resultado ótimo pelo ICA.

Barra	Area (pol²)	Tensão atuante (ksi)
1	0,954	24,3714
2	0,954	18,34382
4	1,764	17,04554
5	1,764	13,41097
6	1,764	10,21923
7	0,111	0,161616
8	0,539	10,52729
9	0,111	0
10	0,440	24,29751
12	0,270	25,00000
14	0,954	19,28398
15	0,111	0

A estrutura ótima encontrada ao fim das 100 execuções do processo de otimizaçao é ilustrada na figura a seguir na Figura 29:

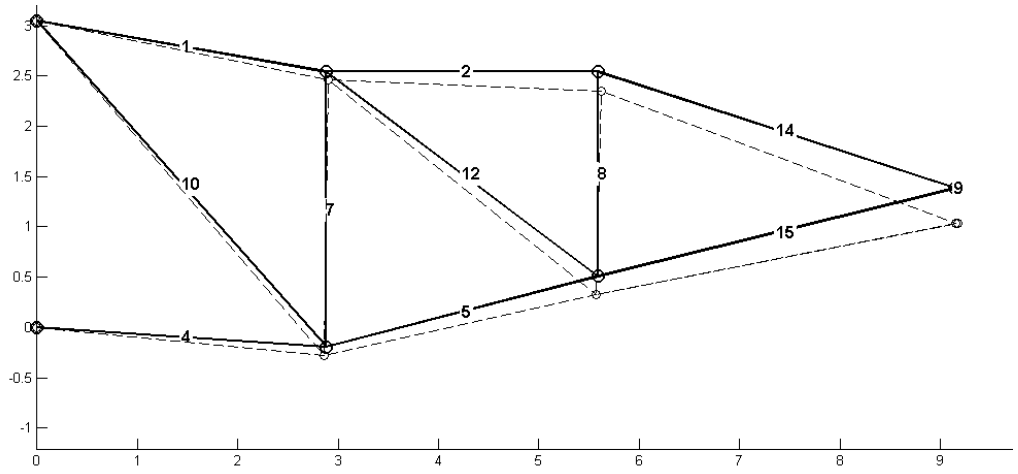
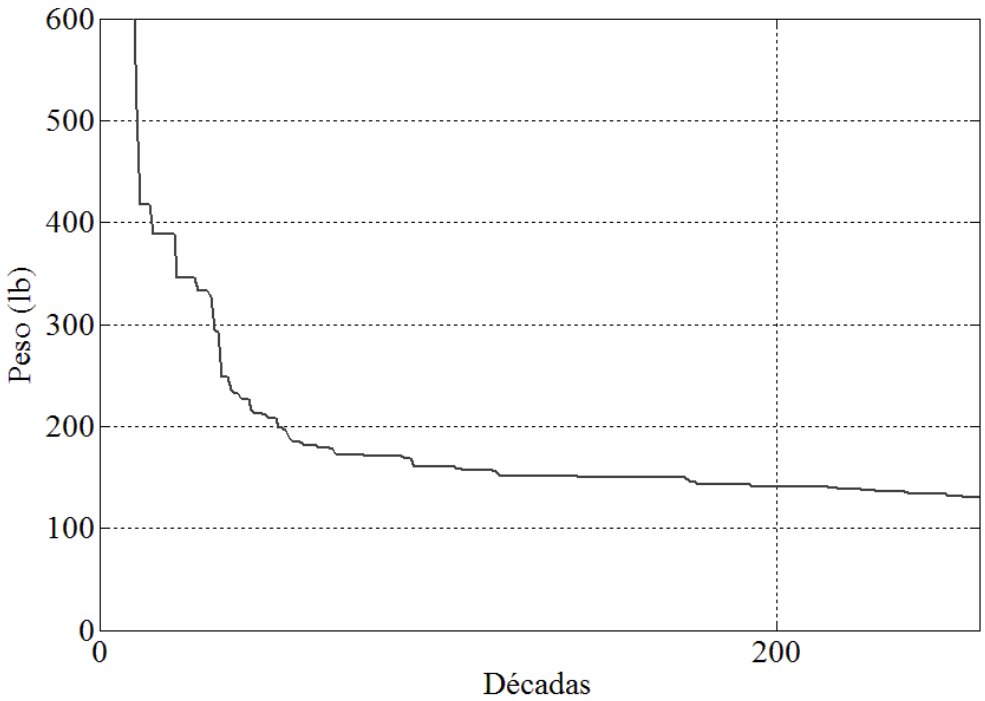


Figura 29 – Resultado ótimo encontrado pelo ICA para o problema 10.3.2

Figura 30 – Convergência típica para o problema 10.3.2 otimizado pelo ICA.



10 CONCLUSÕES

Através deste trabalho pôde-se concluir que problemas mais complexos de otimização são em geral não-convexos e os métodos de otimização metaheurísticos são uma alternativa bem adaptada para lidar com a adversidade imposta por tais problemas. Os métodos metaheurísticos pertencem a um campo de estudos importante e ainda em desenvolvimento, eles apresentam diversas vantagens sobre outros métodos de otimização baseados em gradiente. E apesar de ainda possuírem alguns pontos negativos em aberto, pôde-se observar na literatura acadêmica consultada que a sua utilização é justificada.

Através de uma comparação estatística do uso do ICA com outros algoritmos metaheuristico para a solução de problemas matemáticos, e de engenharia, observou-se que o desempenho do algoritmo proposto foi em geral melhor ou muito próximo dos melhores valores de desempenho encontrados na literatura para os problemas analisados.

Na resolução dos problemas de treliça pôde-se observar que para exemplo 9.3.1 o ICA obteve uma melhor média das avaliações enquanto no 9.3.2 o FA obteve uma média melhor. Tendo em vista esta aparente “afinidade” de cada algoritmo com determinados problemas, podemos assumir que para futuros trabalhos seria oportuno tentar construir uma correlação entre as características do problema sendo otimizado e os parâmetros do algoritmo.

11 REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. J. G. Schoofs. Topics in Applied Mechanics, chapter Structural Optimization History and State-of-theart, pages 339–345. Kluwer Academic Publishers, 1993.

[2]. Coello Coello, Carlos A. Christiansen, Alan D. (2000). Multiobjective optimization of trusses using genetic algorithms. Computers and Structures. Volume 75, issue 6. Pages 647-660.

[3] Galileo Galilei. Dialogues Concerning Two New Sciences. Evanston, Ill. Northwestern University Press, 1950. Publicação original em 1665.

[4] Richardson, James Norman. *Topology Optimization of Truss-like Structures: from Theory to Practice.* 2013. 162 f. Thesis for Doctor in Engineering Sciences - Faculty of Engineering . Brussels School of Engineering, Brussels.

[5] Singiresu S. Rao. ENGINEERING OPTIMIZATION Theory and Practice Third Edition. West Lafayette, Indiana: School of Mechanical Engineering, 1996.

[6]A.O. Griewank, Generalized descent for global optimization, Journal of Optimization Theory and Applications 34 (1) (1981) 11–39.

[7] Heyden, Susanne.Dahlbom, Ola. Olsson, Anders. Sandberg, Göran. *(2005). INTRODUKTION TILL STRUKTURMEKANIKEN. Lund: KFS i Lund AB*

[8] *[Christensen, Peter W. Klarbring, Anders. An Introduction to Structural Optimization. (2008)*]

[9] Kawamura, H. Ohmori, H. Kito N. (2002). Truss topology optimization by a modified genetic algorithm. Structural and Multidisciplinary Optimization. Volume 23, issue 6. Pages 467-473.

[10] Deb, Kalyanmoy. Gulati, Surendra. (2000). Design of Truss-Structures for Minimum Weight using Genetic Algorithms. Finite Elements in Analysis and Design. Volume 37, issue 5. Pages 447-465.

[11] Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to derivative-free optimization. SIAM, Philadelphia (2009).

[12] F. Glover. Future paths for integer programming and links to artificial intelligence. Comp. Oper. Res., 13:533–549, 1986.

[13] C. Blum, A. Roli. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surveys, 35 (3) (2003), pp. 268–308.

[14] Ali Sadollah, Ardeshir Bahreininejad, Hadi Eskandar, and Mohd Hamdi. Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. Applied Soft Computing, 13(5):2592–2612, May 2013.

[15] Dervis Karaboga, Bahriye Akay. A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation 214 (2009) 108–132;

[16] L.F.F. Miguel, R.H. Lopez, L.F.F. Miguel. Multimodal size, shape, and topology optimisation of truss structures usingthe Firefly algorithm. Advances in Engineering Software 56 (2013) 23–37

ANEXOS

ANEXO A – **Algoritmo da Competição Imperialista, sua implementação original como é fornecida pelo autor, em MATLAB:**

```
%% Imperialist Competitive Algorithm (CCA);

% A Socio Politically Inspired Optimization Strategy.
% 2008

% To use this code, you should only prepare your cost function and
apply
% CCA to it. Please read the guide available in my home page.

% Special thank is for my friend Mostapha Kalami Heris whos breadth
wision toward
% artificial intelligence and his programming skils have always been a
% sourse of inspiration for me. He helped me a lot to prepare this
code.
% His email: sm.kalami@gmail.com

% ----------------------------------------
% Esmaeil Atashpaz Gargari
% Control and Intelligent Processing Center of Excellence,
% ECE school of University of Tehran, Iran
% Cellphone: (+98)-932-9011620
% Email: e.atashpaz@ece.ut.ac.ir & atashpaz.gargari@gmail.com
% Home Page: http://www.atashpaz.com

close all
clc; clear
%% Problem Statement
ProblemParams.CostFuncName = 'BenchmarkFunction';    % You should
state the name of your cost function here.
ProblemParams.CostFuncExtraParams = 3;
ProblemParams.NPar = 2;                              % Number of
optimization variables of your objective function. "NPar" is the
dimention of the optimization problem.
ProblemParams.VarMin = -6;                           % Lower limit of
the optimization parameters. You can state the limit in two ways. 1)
2)
ProblemParams.VarMax = 6;                        % Lower limit of the
optimization parameters. You can state the limit in two ways. 1)   2)

% Modifying the size of VarMin and VarMax to have a general form
if numel(ProblemParams.VarMin)==1

ProblemParams.VarMin=repmat(ProblemParams.VarMin,1,ProblemParams.NPar)
;

ProblemParams.VarMax=repmat(ProblemParams.VarMax,1,ProblemParams.NPar)
;
end

ProblemParams.SearchSpaceSize = ProblemParams.VarMax -
ProblemParams.VarMin;

%% Algorithmic Parameter Setting
AlgorithmParams.NumOfCountries = 20;              % Number of initial
countries.
```

```
AlgorithmParams.NumOfInitialImperialists = 3;      % Number of Initial
Imperialists.
AlgorithmParams.NumOfAllColonies = AlgorithmParams.NumOfCountries -
AlgorithmParams.NumOfInitialImperialists;
AlgorithmParams.NumOfDecades = 400;
AlgorithmParams.RevolutionRate = 0.3;              % Revolution is
the process in which the socio-political characteristics of a country
change suddenly.
AlgorithmParams.AssimilationCoefficient = 2;       % In the original
paper assimilation coefficient is shown by "beta".
AlgorithmParams.AssimilationAngleCoefficient = .5;  % In the original
paper assimilation angle coefficient is shown by "gama".
AlgorithmParams.Zeta = 0.02;                       % Total Cost of
Empire = Cost of Imperialist + Zeta * mean(Cost of All Colonies);
AlgorithmParams.DampRatio = 0.99;
AlgorithmParams.StopIfJustOneEmpire = false;         % Use "true" to
stop the algorithm when just one empire is remaining. Use "false" to
continue the algorithm.
AlgorithmParams.UnitingThreshold = 0.02;           % The percent of
Search Space Size, which enables the uniting process of two Empires.

zarib = 1.05;                       % **** Zarib is used to prevent
the weakest impire to have a probability equal to zero
alpha = 0.1;                        % **** alpha is a number in the
interval of [0 1] but alpha<<1. alpha denotes the importance of mean
minimum compare to the global mimimum.

%% Display Setting
DisplayParams.PlotEmpires = false;    % "true" to plot. "false" to
cancel ploting.
if DisplayParams.PlotEmpires
    DisplayParams.EmpiresFigureHandle = figure('Name','Plot of
Empires','NumberTitle','off');
    DisplayParams.EmpiresAxisHandle = axes;
end

DisplayParams.PlotCost = true;    % "true" to plot. "false"
if DisplayParams.PlotCost
    DisplayParams.CostFigureHandle = figure('Name','Plot of Minimum
and Mean Costs','NumberTitle','off');
    DisplayParams.CostAxisHandle = axes;
end

ColorMatrix = [1   0   0  ; 0 1   0    ; 0   0 1    ; 1   1   0  ; 1
0 1    ; 0 1   1    ; 1 1 1       ;
               0.5 0.5 0.5; 0 0.5 0.5  ; 0.5 0 0.5  ; 0.5 0.5 0  ; 0.5
0 0    ; 0 0.5 0    ; 0 0 0.5     ;
               1   0.5 1  ; 0.1*[1 1 1]; 0.2*[1 1 1]; 0.3*[1 1 1];
0.4*[1 1 1]; 0.5*[1 1 1]; 0.6*[1 1 1]];
DisplayParams.ColorMatrix = [ColorMatrix ; sqrt(ColorMatrix)];

DisplayParams.AxisMargin.Min = ProblemParams.VarMin;
DisplayParams.AxisMargin.Max = ProblemParams.VarMax;

%% Creation of Initial Empires
InitialCountries = GenerateNewCountry(AlgorithmParams.NumOfCountries ,
ProblemParams);

% Calculates the cost of each country. The less the cost is, the more
is the power.
```

```
if isempty(ProblemParams.CostFuncExtraParams)
    InitialCost = feval(ProblemParams.CostFuncName,InitialCountries);
else
    InitialCost =
feval(ProblemParams.CostFuncName,InitialCountries,ProblemParams.CostFu
ncExtraParams);
end
[InitialCost,SortInd] = sort(InitialCost);                              %
Sort the cost in assending order. The best countries will be in higher
places
InitialCountries = InitialCountries(SortInd,:);                         %
Sort the population with respect to their cost.

Empires =
CreateInitialEmpires(InitialCountries,InitialCost,AlgorithmParams,
ProblemParams);

%% Main Loop
MinimumCost = repmat(nan,AlgorithmParams.NumOfDecades,1);
MeanCost = repmat(nan,AlgorithmParams.NumOfDecades,1);

if DisplayParams.PlotCost
    axes(DisplayParams.CostAxisHandle);
    if any(findall(0)==DisplayParams.CostFigureHandle)

h_MinCostPlot=plot(MinimumCost,'r','LineWidth',1.5,'YDataSource','Mini
mumCost');
        hold on;

h_MeanCostPlot=plot(MeanCost,'k:','LineWidth',1.5,'YDataSource','MeanC
ost');
        hold off;
        pause(0.05);
    end
end

for Decade = 1:AlgorithmParams.NumOfDecades
    AlgorithmParams.RevolutionRate = AlgorithmParams.DampRatio *
AlgorithmParams.RevolutionRate;

    Remained = AlgorithmParams.NumOfDecades - Decade
    for ii = 1:numel(Empires)
        %% Assimilation;  Movement of Colonies Toward Imperialists
(Assimilation Policy)
        Empires(ii) =
AssimilateColonies(Empires(ii),AlgorithmParams,ProblemParams);

        %% Revolution;  A Sudden Change in the Socio-Political
Characteristics
        Empires(ii) =
RevolveColonies(Empires(ii),AlgorithmParams,ProblemParams);

        %% New Cost Evaluation
        if isempty(ProblemParams.CostFuncExtraParams)
            Empires(ii).ColoniesCost =
feval(ProblemParams.CostFuncName,Empires(ii).ColoniesPosition);
        else
            Empires(ii).ColoniesCost =
feval(ProblemParams.CostFuncName,Empires(ii).ColoniesPosition,ProblemP
arams.CostFuncExtraParams);
```

```
        end

        %% Empire Possession  (****** Power Possession, Empire
Possession)
        Empires(ii) = PossesEmpire(Empires(ii));

        %% Computation of Total Cost for Empires
        Empires(ii).TotalCost = Empires(ii).ImperialistCost +
AlgorithmParams.Zeta * mean(Empires(ii).ColoniesCost);

    end

    %% Uniting Similiar Empires
    Empires =
UniteSimilarEmpires(Empires,AlgorithmParams,ProblemParams);

    %% Imperialistic Competition
    Empires = ImperialisticCompetition(Empires);

    if numel(Empires) == 1 && AlgorithmParams.StopIfJustOneEmpire
        break
    end

    %% Displaying the Results

DisplayEmpires(Empires,AlgorithmParams,ProblemParams,DisplayParams);

    ImerialistCosts = [Empires.ImperialistCost];
    MinimumCost(Decade) = min(ImerialistCosts);
    MeanCost(Decade) = mean(ImerialistCosts);

    if DisplayParams.PlotCost
        refreshdata(h_MinCostPlot);
        refreshdata(h_MeanCostPlot);
        drawnow;
        pause(0.01);
    end

end % End of Algorithm
MinimumCost(end)
```

```
function TheEmpire =
AssimilateColonies(TheEmpire,AlgorithmParams,ProblemParams)

% for i = 1:numel(Imperialists)
%     Imperialists{i}.Number_of_Colonies_matrix =
[Imperialists{i}.Number_of_Colonies_matrix
Imperialists{i}.Number_of_Colonies];
%
%     Imperialists_cost_matrix (i) =
Imperialists{i}.cost_just_by_itself;
%
%     Imperialists_position_matrix(i,:) = Imperialists{i}.position;

NumOfColonies = size(TheEmpire.ColoniesPosition,1);
```

```
Vector = repmat(TheEmpire.ImperialistPosition,NumOfColonies,1)-
TheEmpire.ColoniesPosition;

TheEmpire.ColoniesPosition = TheEmpire.ColoniesPosition + 2 *
AlgorithmParams.AssimilationCoefficient * rand(size(Vector)) .*
Vector;

MinVarMatrix = repmat(ProblemParams.VarMin,NumOfColonies,1);
MaxVarMatrix = repmat(ProblemParams.VarMax,NumOfColonies,1);

TheEmpire.ColoniesPosition=max(TheEmpire.ColoniesPosition,MinVarMatrix
);
TheEmpire.ColoniesPosition=min(TheEmpire.ColoniesPosition,MaxVarMatrix
);
```

```
function z=BenchmarkFunction(x,number)
    if nargin<2
        error('Name or Number of function is not specified.');
    end

    switch number
        case 1
            z=(x'.^2)';
        case 2
            z=zeros(size(x,1),1);
            for i=1:size(x,2);
                z=z+i*x(:,i).^4;
            end
        case 3
            z=0.5+(sin(sqrt(x(:,1).^2+x(:,2).^2))-
0.5)./(1+0.001*(x(:,1).^2+x(:,2).^2)).^2;
        case 4
            z=zeros(size(x,1),1);
            p=ones(size(x,1),1);
            for i=1:size(x,2)
                z=z+1/4000*(x(:,i)-100).^2;
                p=p.*(cos((x(:,i)-100)/sqrt(i))+1);
            end
        case 5
            z=zeros(size(x,1),1);
            for i=1:size(x,2)-1;
                z=z+100*(x(:,i+1)-x(:,i).^2).^2+(x(:,i)-1).^2;
            end
        case 6
            z=sum(x'.^2-10*cos(2*pi*x')+10)';
        otherwise
            error('Invalid function number is used.');
    end

end
```

```
function Empires =
CreateInitialEmpires(InitialCountries,InitialCost,AlgorithmParams,
ProblemParams)
```

```
AllImperialistsPosition =
InitialCountries(1:AlgorithmParams.NumOfInitialImperialists,:);
AllImperialistsCost =
InitialCost(1:AlgorithmParams.NumOfInitialImperialists,:);

AllColoniesPosition =
InitialCountries(AlgorithmParams.NumOfInitialImperialists+1:end,:);
AllColoniesCost =
InitialCost(AlgorithmParams.NumOfInitialImperialists+1:end,:);

if max(AllImperialistsCost)>0
    AllImperialistsPower = 1.3 * max(AllImperialistsCost) -
AllImperialistsCost;
else
    AllImperialistsPower = 0.7 * max(AllImperialistsCost) -
AllImperialistsCost;
end

AllImperialistNumOfColonies =
round(AllImperialistsPower/sum(AllImperialistsPower) *
AlgorithmParams.NumOfAllColonies);
AllImperialistNumOfColonies(end) = AlgorithmParams.NumOfAllColonies -
sum(AllImperialistNumOfColonies(1:end-1));
RandomIndex = randperm(AlgorithmParams.NumOfAllColonies);

Empires(AlgorithmParams.NumOfInitialImperialists).ImperialistPosition
= 0;

for ii = 1:AlgorithmParams.NumOfInitialImperialists
    Empires(ii).ImperialistPosition = AllImperialistsPosition(ii,:);
    Empires(ii).ImperialistCost = AllImperialistsCost(ii,:);
    R = RandomIndex(1:AllImperialistNumOfColonies(ii));
RandomIndex(AllImperialistNumOfColonies(ii)+1:end);
    Empires(ii).ColoniesPosition = AllColoniesPosition(R,:);
    Empires(ii).ColoniesCost = AllColoniesCost(R,:);
    Empires(ii).TotalCost = Empires(ii).ImperialistCost +
AlgorithmParams.Zeta * mean(Empires(ii).ColoniesCost);
end

for ii = 1:numel(Empires)
    if numel(Empires(ii).ColoniesPosition) == 0
        Empires(ii).ColoniesPosition =
GenerateNewCountry(1,ProblemParams);                %
        Empires.ColoniesCost =
feval(ProblemParams.FunctionName,Empires.ColoniesPosition);
    end
end
```

```
function
DisplayEmpires(Empires,AlgorithmParams,ProblemParams,DisplayParams)

    if ~DisplayParams.PlotEmpires
        return;
    end

    if (ProblemParams.NPar ~= 2) && (ProblemParams.NPar ~= 3)
        return;
```

```
    end

    if ~any(findall(0)==DisplayParams.EmpiresFigureHandle)
        return;
    end


    if ProblemParams.NPar == 2
        for ii = 1:numel(Empires)

plot(DisplayParams.EmpiresAxisHandle,Empires(ii).ImperialistPosition(1
),Empires(ii).ImperialistPosition(2),'p',...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',DisplayParams.ColorMatrix(ii,:),...

'MarkerSize',70*numel(Empires(ii).ColoniesCost)/AlgorithmParams.NumOfA
llColonies + 13);
            hold on


plot(DisplayParams.EmpiresAxisHandle,Empires(ii).ColoniesPosition(:,1)
,Empires(ii).ColoniesPosition(:,2),'ok',...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',DisplayParams.ColorMatrix(ii,:),...
                'MarkerSize',8);
        end

        xlim([DisplayParams.AxisMargin.Min(1)
DisplayParams.AxisMargin.Max(1)]);
        ylim([DisplayParams.AxisMargin.Min(2)
DisplayParams.AxisMargin.Max(2)]);
        hold off
    end

    if  ProblemParams.NPar == 3
        figure(1)
        for ii = 1:numel(Empires)

plot3(DisplayParams.EmpiresAxisHandle,Empires(ii).ImperialistPosition(
1),Empires(ii).ImperialistPosition(2),Empires(ii).ImperialistPosition(
3),'p',...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',DisplayParams.ColorMatrix(ii,:),...

'MarkerSize',70*numel(Empires(ii).ColoniesCost)/AlgorithmParams.NumOfA
llColonies + 13);
            hold on


plot3(DisplayParams.EmpiresAxisHandle,Empires(ii).ColoniesPosition(:,1
),Empires(ii).ColoniesPosition(:,2),Empires(ii).ColoniesPosition(:,3),
'ok',...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',DisplayParams.ColorMatrix(ii,:),...
                'MarkerSize',8);
        end

        xlim([DisplayParams.AxisMargin.Min(1)
DisplayParams.AxisMargin.Max(1)]);
```

```
        ylim([DisplayParams.AxisMargin.Min(2)
DisplayParams.AxisMargin.Max(2)]);
        zlim([DisplayParams.AxisMargin.Min(3)
DisplayParams.AxisMargin.Max(3)]);
        hold off
    end

    pause(0.05);
end
```

```
function NewCountry = GenerateNewCountry(NumOfCountries,ProblemParams)

    VarMinMatrix = repmat(ProblemParams.VarMin,NumOfCountries,1);
    VarMaxMatrix = repmat(ProblemParams.VarMax,NumOfCountries,1);
    NewCountry = (VarMaxMatrix - VarMinMatrix) .*
rand(size(VarMinMatrix)) + VarMinMatrix;

end
```

```
function Empires=ImperialisticCompetition(Empires)
    if rand > .11
        return
    end
    if numel(Empires)<=1
        return;
    end

    TotalCosts = [Empires.TotalCost];
    [MaxTotalCost WeakestEmpireInd] = max(TotalCosts);
    TotalPowers = MaxTotalCost - TotalCosts;
    PossessionProbability = TotalPowers / sum(TotalPowers);

    SelectedEmpireInd = SelectAnEmpire(PossessionProbability);

    nn = numel(Empires(WeakestEmpireInd).ColoniesCost);
    jj = myrandint(nn,1,1);

    Empires(SelectedEmpireInd).ColoniesPosition =
[Empires(SelectedEmpireInd).ColoniesPosition

Empires(WeakestEmpireInd).ColoniesPosition(jj,:)];

    Empires(SelectedEmpireInd).ColoniesCost =
[Empires(SelectedEmpireInd).ColoniesCost

Empires(WeakestEmpireInd).ColoniesCost(jj)];

    Empires(WeakestEmpireInd).ColoniesPosition =
Empires(WeakestEmpireInd).ColoniesPosition([1:jj-1 jj+1:end],:);
    Empires(WeakestEmpireInd).ColoniesCost =
Empires(WeakestEmpireInd).ColoniesCost([1:jj-1 jj+1:end],:);

    %% Collapse of the the weakest colony-less Empire
    nn = numel(Empires(WeakestEmpireInd).ColoniesCost);
    if nn<=1
```

```
        Empires(SelectedEmpireInd).ColoniesPosition =
[Empires(SelectedEmpireInd).ColoniesPosition

Empires(WeakestEmpireInd).ImperialistPosition];

        Empires(SelectedEmpireInd).ColoniesCost =
[Empires(SelectedEmpireInd).ColoniesCost

Empires(WeakestEmpireInd).ImperialistCost];

        Empires=Empires([1:WeakestEmpireInd-1
WeakestEmpireInd+1:end]);
    end

end

function Index = SelectAnEmpire(Probability)
    R = rand(size(Probability));
    D = Probability - R;
    [MaxD Index] = max(D);
end
```

```
function y = myrandint(MaxInt,m,n)
%function y = myrandint(MaxInt,m,n)
% This functions creates random numbers between [1 MaxInt] (1 itself
and MaxInt itself)
if nargin == 1
    y = ceil(rand*MaxInt);
elseif nargin == 3
    y = ceil(rand(m,n)*MaxInt);
else
    warning('Incorrect Number of Inputs');
end
```

```
function TheEmpire = PossesEmpire(TheEmpire)

    ColoniesCost = TheEmpire.ColoniesCost;

    [MinColoniesCost BestColonyInd]=min(ColoniesCost);
    if MinColoniesCost < TheEmpire.ImperialistCost

        OldImperialistPosition = TheEmpire.ImperialistPosition;
        OldImperialistCost = TheEmpire.ImperialistCost;

        TheEmpire.ImperialistPosition =
TheEmpire.ColoniesPosition(BestColonyInd,:);
        TheEmpire.ImperialistCost =
TheEmpire.ColoniesCost(BestColonyInd);

        TheEmpire.ColoniesPosition(BestColonyInd,:) =
OldImperialistPosition;
        TheEmpire.ColoniesCost(BestColonyInd) = OldImperialistCost;
    end
```

```
end
```

```
function TheEmpire =
RevolveColonies(TheEmpire,AlgorithmParams,ProblemParams)
    NumOfRevolvingColonies = round(AlgorithmParams.RevolutionRate *
numel(TheEmpire.ColoniesCost));
    RevolvedPosition = GenerateNewCountry(NumOfRevolvingColonies ,
ProblemParams);
    R = randperm(numel(TheEmpire.ColoniesCost));
    R = R(1:NumOfRevolvingColonies);
    TheEmpire.ColoniesPosition(R,:) = RevolvedPosition;
end
```

```
function
Empires=UniteSimilarEmpires(Empires,AlgorithmParams,ProblemParams)

    ThereshholdDistance = AlgorithmParams.UnitingThreshold *
norm(ProblemParams.SearchSpaceSize);
    NumOfEmpires = numel(Empires);

    for ii = 1:NumOfEmpires-1
        for jj = ii+1:NumOfEmpires
            DistanceVector = Empires(ii).ImperialistPosition -
Empires(jj).ImperialistPosition;
            Distance = norm(DistanceVector);
            if Distance<=ThereshholdDistance
                if Empires(ii).ImperialistCost <
Empires(jj).ImperialistCost
                    BetterEmpireInd=ii;
                    WorseEmpireInd=jj;
                else
                    BetterEmpireInd=jj;
                    WorseEmpireInd=ii;
                end

                Empires(BetterEmpireInd).ColoniesPosition =
[Empires(BetterEmpireInd).ColoniesPosition

Empires(WorseEmpireInd).ImperialistPosition

Empires(WorseEmpireInd).ColoniesPosition];

                Empires(BetterEmpireInd).ColoniesCost =
[Empires(BetterEmpireInd).ColoniesCost

Empires(WorseEmpireInd).ImperialistCost

Empires(WorseEmpireInd).ColoniesCost];

                % Update TotalCost for new United Empire
                Empires(BetterEmpireInd).TotalCost =
Empires(BetterEmpireInd).ImperialistCost + AlgorithmParams.Zeta *
mean(Empires(BetterEmpireInd).ColoniesCost);

                Empires = Empires([1:WorseEmpireInd-1
WorseEmpireInd+1:end]);
```

```
                return;
            end

        end
    end

end
```

ANEXO B – **Implementações feita pelo autor deste trabalho para a otimização dos problemas propostos.**

Foram adicionadas ao código original apenas uma rotina para a avaliação de eficiência (eficiencia.m), responsável por executar a otimização duzentas vezes para os fins estatísticos. E os códigos responsáveis pelo cálculo das funções objetivo e respectivas penalizações dos problemas propostos.
Portanto, apenas a rotina principal alterada do ICA será exibida no Anexo B. As demais rotinas do algoritmo que não sofreram alterações não serão repetidas aqui.

```
%% Eficiência do algoritmo.

clear
vezes=200;
x=zeros(vezes,1);
for i=1:vezes

u='griewank'; %substituir termo entre apostrofes pelo problema a ser
analisado:
              %"griewank", "shekel", "vigasolda", "spring",
"tenbartruss", "fifteenbartruss"

    x(i,1) =
ImperialistCompetitveAlgorithm_GlobalOptimizationStrategy(u);
end

media=mean(x)
deviopadrao=std(x)
minimo=min(x)
maximo=max(x)
```

Algoritmo ICA adaptado para resolver os problemas propostos:

```
function [l] =
ImperialistCompetitveAlgorithm_GlobalOptimizationStrategy(problema)

 close all
clc; clear
%% Problem Statement
ProblemParams.CostFuncName = 'problema';  % You should state the name
of your cost function here.
```

```
%alterar parâmetros abaixo de acordo com o problema sendo otimizado
ProblemParams.NPar = 30;       % Number of optimization variables of
your objective function. "NPar" is the dimention of the optimization
problem.
ProblemParams.VarMin = -600;   % Lower limit of the optimization
parameters. You can state the limit in two ways. 1)   2)
ProblemParams.VarMax = 600;    % Lower limit of the optimization
parameters. You can state the limit in two ways. 1)   2)

% Modifying the size of VarMin and VarMax to have a general form
if numel(ProblemParams.VarMin)==1

ProblemParams.VarMin=repmat(ProblemParams.VarMin,1,ProblemParams.NPar)
;

ProblemParams.VarMax=repmat(ProblemParams.VarMax,1,ProblemParams.NPar)
;
end

ProblemParams.SearchSpaceSize = ProblemParams.VarMax -
ProblemParams.VarMin;

%% Algorithmic Parameter Setting
AlgorithmParams.NumOfCountries = 800;               % Number of
initial countries.
AlgorithmParams.NumOfInitialImperialists = 15;      % Number of
Initial Imperialists.
AlgorithmParams.NumOfAllColonies = AlgorithmParams.NumOfCountries -
AlgorithmParams.NumOfInitialImperialists;
AlgorithmParams.NumOfDecades =625;
AlgorithmParams.RevolutionRate = 0.9;               % Revolution is
the process in which the socio-political characteristics of a country
change suddenly.
AlgorithmParams.AssimilationCoefficient = 1.00;        % In the
original paper assimilation coefficient is shown by "beta".
AlgorithmParams.AssimilationAngleCoefficient = 4;  % In the original
paper assimilation angle coefficient is shown by "gama".
AlgorithmParams.Zeta = 99;                          % Total Cost of
Empire = Cost of Imperialist + Zeta * mean(Cost of All Colonies);
AlgorithmParams.DampRatio = 0.99;
AlgorithmParams.StopIfJustOneEmpire = false;         % Use "true" to
stop the algorithm when just one empire is remaining. Use "false" to
continue the algorithm.
AlgorithmParams.UnitingThreshold = 0.0003;            % The percent of
Search Space Size, which enables the uniting process of two Empires.

zarib = 1.05;                       % **** Zarib is used to prevent
the weakest impire to have a probability equal to zero
alpha = 0.1;                        % **** alpha is a number in the
interval of [0 1] but alpha<<1. alpha denotes the importance of mean
minimum compare to the global mimimum.

%% Display Setting
DisplayParams.PlotEmpires = false;    % "true" to plot. "false" to
cancel ploting.
if DisplayParams.PlotEmpires
    DisplayParams.EmpiresFigureHandle = figure('Name','Plot of
Empires','NumberTitle','off');
    DisplayParams.EmpiresAxisHandle = axes;
```

```
end

DisplayParams.PlotCost = false;    % "true" to plot. "false"
if DisplayParams.PlotCost
    DisplayParams.CostFigureHandle = figure('Name','Plot of Minimum
and Mean Costs','NumberTitle','off');
    DisplayParams.CostAxisHandle = axes;
end

ColorMatrix = [1   0   0  ; 0 1   0    ; 0   0 1    ; 1   1   0  ; 1
0 1    ; 0 1   1    ; 1 1 1       ;
               0.5 0.5 0.5; 0 0.5 0.5  ; 0.5 0 0.5  ; 0.5 0.5 0  ; 0.5
0 0    ; 0 0.5 0    ; 0 0 0.5     ;
               1   0.5 1  ; 0.1*[1 1 1]; 0.2*[1 1 1]; 0.3*[1 1 1];
0.4*[1 1 1]; 0.5*[1 1 1]; 0.6*[1 1 1]];
DisplayParams.ColorMatrix = [ColorMatrix ; sqrt(ColorMatrix)];

DisplayParams.AxisMargin.Min = ProblemParams.VarMin;
DisplayParams.AxisMargin.Max = ProblemParams.VarMax;

%% Creation of Initial Empires
InitialCountries = GenerateNewCountry(AlgorithmParams.NumOfCountries ,
ProblemParams);

% Calculates the cost of each country. The less the cost is, the more
is the power.
if isempty(ProblemParams.CostFuncExtraParams)
    InitialCost = feval(ProblemParams.CostFuncName,InitialCountries);
else
    InitialCost = feval(ProblemParams.CostFuncName,InitialCountries);
end
[InitialCost,SortInd] = sort(InitialCost);                          %
Sort the cost in assending order. The best countries will be in higher
places
InitialCountries = InitialCountries(SortInd,:);                     %
Sort the population with respect to their cost.

Empires =
CreateInitialEmpires(InitialCountries,InitialCost,AlgorithmParams,
ProblemParams);

%% Main Loop
MinimumCost = repmat(nan,AlgorithmParams.NumOfDecades,1);
MeanCost = repmat(nan,AlgorithmParams.NumOfDecades,1);

if DisplayParams.PlotCost
    axes(DisplayParams.CostAxisHandle);
    if any(findall(0)==DisplayParams.CostFigureHandle)

h_MinCostPlot=plot(MinimumCost,'r','LineWidth',1.5,'YDataSource','Mini
mumCost');
        hold on;

h_MeanCostPlot=plot(MeanCost,'k:','LineWidth',1.5,'YDataSource','MeanC
ost');
        hold off;
        pause(0.05);
    end
end
```

```
for Decade = 1:AlgorithmParams.NumOfDecades


    Remained = AlgorithmParams.NumOfDecades - Decade;
    for ii = 1:numel(Empires)
        %% Assimilation;  Movement of Colonies Toward Imperialists
(Assimilation Policy)
        Empires(ii) =
AssimilateColonies(Empires(ii),AlgorithmParams,ProblemParams);

        %% Revolution;  A Sudden Change in the Socio-Political
Characteristics
        Empires(ii) =
RevolveColonies(Empires(ii),AlgorithmParams,ProblemParams);

        %% New Cost Evaluation
        if isempty(ProblemParams.CostFuncExtraParams)
            Empires(ii).ColoniesCost =
feval(ProblemParams.CostFuncName,Empires(ii).ColoniesPosition);
        else
            Empires(ii).ColoniesCost =
griewank(Empires(ii).ColoniesPosition,ProblemParams.NPar);
        end

        %% Empire Possession  (****** Power Possession, Empire
Possession)
        Empires(ii) = PossesEmpire(Empires(ii));

        %% Computation of Total Cost for Empires
        Empires(ii).TotalCost = Empires(ii).ImperialistCost +
AlgorithmParams.Zeta * mean(Empires(ii).ColoniesCost);

    end

    %% Uniting Similiar Empires
    Empires =
UniteSimilarEmpires(Empires,AlgorithmParams,ProblemParams);

    %% Imperialistic Competition
    Empires = ImperialisticCompetition(Empires);

    if numel(Empires) == 1 && AlgorithmParams.StopIfJustOneEmpire
        break
    end

    %% Displaying the Results


DisplayEmpires(Empires,AlgorithmParams,ProblemParams,DisplayParams);

    ImerialistCosts = [Empires.ImperialistCost];
    MinimumCost(Decade) = min(ImerialistCosts);
    MeanCost(Decade) = mean(ImerialistCosts);

    if DisplayParams.PlotCost
        refreshdata(h_MinCostPlot);
        refreshdata(h_MeanCostPlot);
        drawnow;
        pause(0.00);
```

```
    end
     ImerialistCosts = [Empires.ImperialistCost];

fprintf('ICA|%5.0f -----> %9.16f  %5.0f
IMPÉRIOS\n',Decade,min(ImerialistCosts), numel(ImerialistCosts));
end % End of Algorithm
MinimumCost(end)
l = MinimumCost(end);

end
```

Função de Griewank:

```
function [y] = griewank(xx)



for k=1:size(xx,1)
d = length(xx(k,:));
sum = 0;
prod = 1;

for ii = 1:d
    xi = xx(k,ii);
    sum = sum + xi^2/4000;
    prod = prod * cos(xi/sqrt(ii));
end

y(k) = sum - prod + 1;

end
y=y';
end
```

Rotina para desenhar a função de Griewank:

```
clear
x0=[-50:0.5:50];
y0=x0;
[X0,Y0]=meshgrid(x0,y0);
[row,col]=size(X0);
for l=1:col
    for h=1:row
        z(h,l)=griewank([X0(h,l),Y0(h,l)]);
    end
end
hold on
contourf(X0,Y0,z,10,'EdgeColor','none')
surf(X0,Y0,z)
hold off
```

Função de Shekel:

```
function y = shekel(x)


w = size(x,1);
y = zeros(w,1);
for z= 1:w;

m = 10;
a = ones(10,2);
a(1,:) = 4.0*a(1,:);
a(2,:) = 1.0*a(2,:);
a(3,:) = 8.0*a(3,:);
a(4,:) = 6.0*a(4,:);
a(5,:) = 6.0*a(5,:);
for j = 1:2;
   a(5,2*j-1) = 3.0; a(5,2*j) = 7.0;
   a(6,2*j-1) = 2.0; a(6,2*j) = 9.0;
   a(7,j)     = 5.0; a(7,j+2) = 3.0;
   a(8,2*j-1) = 8.0; a(8,2*j) = 1.0;
   a(9,2*j-1) = 6.0; a(9,2*j) = 2.0;
   a(10,2*j-1)= 7.0; a(10,2*j)= 3.6;
end
c(1) = 0.1; c(2) = 0.2; c(3) = 0.2; c(4) = 0.4; c(5) = 0.4;
c(6) = 0.6; c(7) = 0.3; c(8) = 0.7; c(9) = 0.5; c(10)= 0.5;
s = 0;
for j = 1:m;
   p = 0;
   for i = 1:5
      p = p+(x(z,i)-a(j,i))^2;
   end
   s = s+1/(p+c(j));
end

y(z) = -s;


end
```

Função objetivo do problema da Viga Soldada

```
function [z,x]=solda(x)


for i=1:size(x,1)

if x(i,1)<0.125
    x(i,1)=0.125;
end

if x(i,1)>5
    x(i,1)=5;
end
```

```
if x(i,2)<0.1
    x(i,2)=0.1;
end
if x(i,2)>10
    x(i,2)=10;
end

if x(i,3)<0.1
   x(i,3)=0.1;
end
if x(i,3)>10
    x(i,3)=10;
end

if x(i,4)<0.1
    x(i,4)=0.1;
end
if x(i,4)>2
    x(i,4)=2;
end

P = 6000;
L = 14;
E = 30e+6;
G = 12e+6;
t_max = 13600;
s_max = 30000;
d_max = 0.25;
M = P*(L+x(i,2)/2);
R = sqrt(0.25*(x(i,2)^2)+((x(i,1)+x(i,3))/2)^2);
J = 2*(sqrt(2)*x(i,1)*x(i,2)*(x(i,2)^2/12+0.25*(x(i,1)+x(i,3))^2));
P_c = (4.013*E*sqrt(x(i,3)^2*x(i,4)^6/36))/L^2*(1-
(x(i,3)/(2*L))*sqrt(E/(4*G)));
t1 = P/(sqrt(2)*x(i,1)*x(i,2));
t2 = M*R/J;
t = sqrt(t1^2+t1*t2*x(i,2)/R+t2^2);
s = (6*P*L)/(x(i,4)*x(i,3)^2);
d = 4*P*L^3/(E*x(i,4)*x(i,3)^3);


%Restrições
g=[0 0 0 0 0];

g(1) = (t-t_max);
g(2) = (s-s_max);
g(3) = (x(i,1)-x(i,4));
g(4) = 1.10471*x(i,1)^2*x(i,2)+0.04811*x(i,3)*x(i,4)*(14.0+x(i,2))-5;
g(5) = 0.125-x(i,1);
g(6) = (d-d_max);
g(7) = (P-P_c);

%Penalização com coeficientes de penalização obtidos de maneira
empírica
delta(1)=max(0,g(1,1)*7);
delta(2)=max(0,g(1,2)*0.1);
delta(3)=max(0,g(1,3)*5);
delta(4)=max(0,g(1,4)*0.1);
delta(5)=max(0,g(1,5)*1);
delta(6)=max(0,g(1,6)*0.5);
delta(7)=max(0,g(1,7)*0.5);
```

```
delta
P=sum(delta);



%Função custo

z(i,1)= 1.10471*x(i,1)^2*x(i,2)+0.04811*x(i,3)*x(i,4)*(14.0+x(i,2))+P;


end
end
```

Função objetivo par ao problema *tension/compression spring*:

```
function [z,x]=spring(x)

for i=1:size(x,1)
if x(i,1)<0.05
   x(i,1)=0.05;
end

if x(i,1)>2
    x(i,1)=2;
end

if x(i,2)<0.25
    x(i,2)=0.25;
end
if x(i,2)>1.3
    x(i,2)=1.3;
end

if x(i,3)<2
   x(i,3)=2;
end
if x(i,3)>15
    x(i,3)=15;
end

%restrições

g(1,1) = 1-(x(i,2)^3*x(i,3))/(71785*x(i,1)^4);
g(1,2) = (4*x(i,2)^2-x(i,1)*x(i,2))/(12566*x(i,1)^3*(x(i,2)-
x(i,1)))+1/(5108*x(i,1)^2)-1;
g(1,3) = 1-140.45*x(i,1)/(x(i,3)*x(i,2)^2);
g(1,4) = (x(i,1)+x(i,2))/1.5-1;

%penalizações com coeficientes obtidos de maneira empírica
delta(1,1)=max(0,g(1,1)*0.05);
delta(1,2)=max(0,g(1,2)*0.05);
delta(1,3)=max(0,g(1,3)*0.05);
delta(1,4)=max(0,g(1,4)*0.05);
```

```
F=sum(delta(1,:));

z(i,1) = x(i,1)^2*x(i,2)*(x(i,3)+2)+F;

end
end
```