INTRODUÇÃO A ALGORITMOS GENÉTICOS

ANDRÉ FIGUEIRA& DICKSIANO MELO[1]

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Algoritmo genético é um tipo de algoritmo de optimização, ou seja seu uso é destinado a encontrar máximo e mínimos de funções. Este projeto se

resume à uma introdução deste tipo de algoritmo, à explicação de como se constrói um e como este é utilizado para resolver problemas.

1 Introdução

Algoritmos Genéticos pertencem ao grupo dos algoritmos de otimização, ergo são usados para encontrar soluções otimizadas para um dado problema computacional que maximiza ou minimiza uma função em particular. Representam um ramo de campo de estudo chamado computação evolucionária, que se baseia nos processos biológicos de reprodução e seleção natural para encontrar soluções mais adequadas. Como na evolução, muitos dos processos que regem estes algoritmos são aleatórios, mas há possibilidade de ajustar os níveis de controle e aleatoriedade. Estes algoritmos são mais poderosos e eficientes que uma busca aleatória e algoritmos de busca exaustivos, ainda assim requirindo nenhuma informação adicional sobre o problema. São capazes de encontrar respostas que outros métodos não podem devido a falta de continuidade, derivadas, linearidades e outras características.[1]

2 Componentes, Estrutura e Terminologia

Como algoritmos genéticos espelham um processo biológico, a terminologia envolvida também rouba desta ciência. Basicamente, os componentes comuns entre estes algoritmos são:

- Uma função de adaptação "*fitness*" para ser otimizada
- Uma população de indivíduos
- Uma função de reprodução "*crossover*" para produzir a próxima geração de indivíduos.
- Uma função de mutação para indivíduos de uma nova geração.
- Uma função de seleção para selecionar os indivíduos mais adaptados para a próxima geração.

2.1 População e indivíduos

Um indivíduo se refere à um candidato à solução, por exemplo se o problema é encontrar o máximo de uma função $f(x)$, $x = 3$ seria por exemplo um indivíduo. Uma população seria um conjunto de indivíduos. Um indivíduo, todavia, não carrega sua informação diretamente mas sim por meio de uma cadeia de dados, como se fosse um código genético. Para representações numéricas pode-se utilizar uma cadeia binária, então para $x = 3$ e considerando uma cadeia de tamanho 4, o código genético seria:

$$[0011]$$

.

* *Instituo Tecnológico de Aeronáutica*

2.1.1 Tamanho e Dimensão

Tamanho se refere ao número de termos que a cadeia binária possui, quanto maior o tamanho maior precisão é possível obter. Ás vezes deseja-se que um indivíduo acomode um código genético em partes e que estas partes sejam independentes entre si. É o caso por exemplo de otimizar uma função de duas variáveis $f(x, y)$, neste caso o código genético de um indivíduo deve se reservar às duas variáveis separadamente, por exemplo para $X = (3, 1)$ e tamanho 4, o código seria:

$$[0011\ 0001]$$

Ao número de partes se nomeia dimensões, um problema de duas variáveis possui então duas dimensões.

2.1.2 Valor de adaptação

Cada indivíduo carrega consigo um valor de adaptação, se x é a informação armazenada pelo indivíduo, o seu valor de adaptação é $f(x)$ sendo f a função de adaptação.

2.2 Crossing-Over

O processo de *crossing-over* é o responsável pela variabilidade genética, novos indivíduos são formados por meio da permutação dos genes de indivíduos já existentes. Como exemplo, eis dois indivíduos:

$$Individuo\ 1 : [0100]$$

$$Individuo\ 2 : [1000]$$

O ponto de *crossing-over* é escolhido aleatoriamente, mas supondo que este seja após o primeiro termo, dois filhos podem ser formados, um com o primeiro termo do Indivíduo 1 e os três últimos termos do Indivíduo 2 e o outro com o primeiro termo do Indivíduo 2 e os três últimos termos do Indivíduo 1:

$$Filho\ 1 : [0000]$$

$$Filho\ 2 : [1100]$$

2.3 Mutação

Suponha que a resposta de um problema seja a cadeia:

$$[1111]$$

Uma população original é formada aleatoriamente, suponha que uma população original de 4 indivíduos tenha assim se formado:

$$[1110]$$

$$[0100]$$

$$[0000]$$

$$[1100]$$

Note que o gene 1 no quarto termo não existe em nenhum indivíduo desta população, então, se por meios somente da função de reprodução, a resposta

ideal jamais seria alcançada. A mutação é necessária para atingir o valor ideal ao invés de uma solução "razoável". De maneira mais simples, a mutação possui uma probabilidade C de ocorrer para cada indivíduo a cada geração. Se acontecer, um gene aleatório é escolhido para ser invertido. Por exemplo, seja o indivíduo :

$$[1110]$$

Uma mutação no quarto gene faria este indivíduo se tornar em:

$$[1111]$$

.

2.4 Seleção

A função de seleção age como a natureza no processo de seleção natural, ela escolhe os indivíduos mais adaptados para seguir na próxima geração desprezando uma parcela da população original. O selecionamento ocorre sob critério do valor de adaptação, $f(x)$.

3 Aplicações

3.1 Encontrar máximos, mínimos e zeros de funções

A primeira aplicação é diretamente matemática, dada uma função de n variáveis, encontrar um máximo ou mínimo global dentro de um certo domínio. Considere então a função $f(X)$ em que X representa um vetor de D variáveis limitadas à um certo domínio. A aplicação do algoritmo será explicado pelas seguintes partições:

1. Criação-Codificação
2. Decodificação
3. Cross-Over
4. Mutação
5. Seleção
6. Programa principal

3.1.1 Criação-Codificação

Nesta parte da implementação, uma população aleatória é criada a não ser por três critérios:

1. Número inicial de indivíduos (N)
2. Tamanho de cada dimensão de cada indivíduo (T)
3. Dimensão de cada indivíduo. (D)

Então, por exemplo, para $N = 3$, $T = 8$ e $D = 2$ uma possível população seria:

$$\mathit{Individuo}\,1\ [01100110\ 11100001]$$

$$\mathit{Individuo}\,2\ [01111110\ 00101111]$$

$$\mathit{Individuo}\,3\ [01000010\ 11100111]$$

A informação contida nestas cadeias será concebida na próxima partição.

3.1.2 Decodificação

Cada cadeia contém um número decimal encriptado, é preciso uma função para decodificar este valor. Sendo a variável limitada da seguinte forma:

$$a < x < b$$

, V_x o valor decimal da cadeia binária e V_m o valor decimal máximo que uma cadeia de mesmo tamanho possa assumir, então o valor de decodificação é dado pela fórmula:

$$\frac{V_x}{V_m} \times (b - a) + a$$

Duas observações devem ser feitas:

1. Quando se diz cadeia, a referência é à uma cadeia de uma dimensão, indivíduos de D dimensões possuirão D valores de decodificação, sendo que cada dimensão possui seus próprios intervalos de domínio.
2. Os valores que as variáveis podem assumir são discretizados pelo tamanho T de cada dimensão, quanto maior T mais valores de x são possíveis e por conseguinte maior será a precisão.

Após encontrado os valores decimais, resta marcar o indivíduos com o seu valor de adaptação, valor "*fitness*". Um termo a mais é reservado à cadeia binária para armazenar este valor. Segue um exemplo, seja a função de estudo a função:

$$f(x, y) = 1 - x^2 - y^2$$

para:

$$-1 < x < 1 \quad 0 < y < 1$$

e o seguinte indivíduo:

$$[0001\ 0110]$$

Usando a fórmula:

$$x = \frac{2^0}{2^4 - 1} \times (1 - (-1)) + -1 = -0.8667$$

$$y = \frac{2^2 + 2^1}{2^4 - 1} \times (1 - 0) + 0 = 0.400$$

O valor de adaptação é então:

$$f(-0.8667, 0.400) = 0.0888$$

O vetor linha de tal indivíduo, após a decodificação, enfim termina assim:

$$[0001\ 0110 \quad 0.0888]$$

3.1.3 Cross-Over

A função de reprodução se consiste em formar novos indivíduos adotando como base os já existentes. Na implementação, os indivíduos novos não substituem os já existentes e sim se adicionam à população, a morte de indivíduos é papel da função de seleção. Não há muito o que detalhar aqui que já não foi descrito na explicação geral, dois pais sãos escolhidos então um ponto de cruzamento é selecionado aleatoriamente, então as partes das cadeias permutam formando dois novos indivíduos (os filhos). Se o indivíduo possui mais de uma dimensão, o cruzamento ocorre somente entre as

dimensões em pontos de cruzamento diferentes,. O único detalhe que sobra é explicitar como ocorre a seleção dos pais. Na implementação, metade dos pais é escolhido aleatoriamente e a outra é a própria população atual, garantido que todos pertençam a ao menos um pareamento, sem distinção de repetição.

3.1.4 Mutação

Novamente, não há muito a adicionar. Definida uma chance de mutação C, todo indivíduo em toda rodada possui tal chance de mutar, evento que se consiste na reversão aleatória de algum gene de seu código (0 se torna 1 e 1 se torna 0). Se o indivíduo possui mais de uma dimensão, todas as suas dimensões mutarão, porém em posições diferentes.

3.1.5 Seleção

Existem várias formas de implementar uma seleção, a utilizada é bem simples, escolher a metade da população mais adaptada, ou seja escolher os N/2 indivíduos com o maior valor de adaptação para seguir na população enquanto os outros são esquecidos. Uma observação deve ser feita aqui, como a função de reprodução triplica a população e esta função apenas a corta pela metade, a população aumenta com o progredir das gerações. Para limitar este crescimento, se a população atingir um valor maior que dez vezes o número original, acontece uma seleção catastrófica que deixa apenas um quinto da população atual viva. (um dilúvio).

3.1.6 Programa principal

Primeiro se cria a população com 1, atribui-se valores de adaptação para cada indivíduo com 2. Em seguida executa-se uma seleção com 5. A partir daqui as iterações repetem os passos $3-4-2-5$. A execução do código termina em duas ocasiões:

1. O número máximo de iterações é atingido
2. O maior valor de adaptação da população não se altera por 3 gerações seguidas.

3.2 Problema do Caixeiro Viajante

O **Problema do Caixeiro Viajante** (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Este é um dos problemas de otimização combinatórias mais conhecidos e é aplicados em várias áreas, até mesmo no planejamento de trajetórias de robôs autônomos. Ele é um problema de otimização **NP-difícil** inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.[2]

Formulação: Suponha que um caixeiro viajante tenha de visitar n cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas e que de cada uma delas pode-se ir diretamente a qualquer outra. O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total. Assumimos, também, que o problema é simétrico, ou seja, a $d_{AB} = d_{BA}$.

Por exemplo, se tivermos quatro cidades **A, B, C e D**, uma rota que o caixeiro deve considerar poderia ser: saia de A e daí vá para B, dessa vá para C, e daí vá para D e então volte a A. Quais são as outras possibilidades ? É muito fácil ver que existem seis rotas possíveis:

ABCDA

ABDCA

ACBDA

ACDBA

ADBCA

ADCBA

A solução do problema consiste em encotrar qual rota possui menor caminho percorrido. A aplicação do algoritmo será explicado pelas seguintes partições:

1. Criação-Codificação
2. Decodificação
3. Cross-Over
4. Mutação
5. Seleção
6. Programa principal

3.2.1 Criação-Codificação

Nesta parte da implementação, uma população aleatória é criada a não ser por dois critérios:

1. Número cidades (Ncidades)
2. Número de cromossomos da população (Ncromossomos)

Então, por exemplo, para Ncidades = 4, Ncidades = 4, uma possível população seria:

Cromossomo 1 [1 2 3 4]

Cromossomo 2 [3 2 1 4]

Cromossomo 3 [2 4 3 1]

Cromossomo 4 [4 2 1 3]

3.2.2 Decodificação

Cada cadeia contém a ordem de visita das cidades. Por exemplo:

Cromossomo 1 [1 2 3 4]

Indica que o caixeiro visita $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$.

Observe que os casos [1 2 3 4] e [4 3 2 1] são equivalentes, pois o que importa é o somatório das distâncias entre as cidades.

3.2.3 Cross-Over

A a operação de Cross-Over definida para Algoritmo Genético binário não funciona no caso do Problema do Caixeiro Viajante, pois cada cidade deve aparecer uma única vez.

Cromossomo 1 [3 5 ∗ 1 2 4]

Cromossomo 2 [1 4 ∗ 5 3 2]

Após o Cross-Over:

Cromossomo 1 [3 5 ∗ **5 3** 2]

Cromossomo 2 [1 4 ∗ **1** 2 **4**]

Observe que os cromossomos resultantes são inválidos, pois existem cidades que aparecem mais de uma vez e algumas não aprecem nenhuma vez. Dessa forma, será necessário adaptar o processo de Cross-Over utilizando um método denominado **Cycle Crossover**.

Cromossomo 1 [4 1 5 ∗3 ∗ 2 6]

Cromossomo 2 [3 4 6 ∗ 2 ∗ 1 5]

Cross-over:

Cromossomo 1 [4 1 5 ∗2∗ 2 6]

Cromossomo 2 [3 4 6 ∗3∗ 1 5]

Cromossomo 1 [4 1 5 2 ∗1∗ 6]

Cromossomo 2 [3 4 6 3 ∗2∗ 5]

Cromossomo 1 [4 ∗4∗ 5 2 1 5]

Cromossomo 2 [3 ∗1∗ 6 3 2 6]

Cromossomo 1 [∗3∗ 4 5 2 1 5]

Cromossomo 2 [∗4∗ 1 6 3 2 6]

Inicialmente, é escolhida uma posição aleatória do cromossomo. As posições marcadas por * são trocadas. Após a primeira troca, haverão cidades repetidas. Assim, deve-se trocar os valores repetidos até que não hajam valores repetidos.

3.2.4 Mutação

Basta escolher duas cidades de um cromossomo e trocá-los. A probabilidade de ocorrer mutação é definido pela variável **taxaDeMutacao**.

Cromossomo [∗4∗ 1 5 ∗3∗ 2 6]

Cromossomo [∗3∗ 1 5 ∗4∗ 2 6]

3.2.5 Seleção

Existem várias formas de implementar uma seleção, a utilizada é bem simples, escolher a metade da população mais adaptada, ou seja escolher os N/2 indivíduos com o maior valor de adaptação para seguir na população enquanto os outros são esquecidos.

3.2.6 Programa principal

Inicialmente, se cria a população inicial contendo N cromossomos. Em seguida, calcula-se o valor de adaptação de cada cromossomo utilizando a função **funcaoFitness**. A partir daqui, realiza-se Cross-Over e Mutação para cada geração, sempre selecionando a metade da população mais bem adaptada. A execução do código termina quando o número máximo de iterações é atingido. A solução do problema é o cromossomo com menor valor de função fitness.

4 Manual do usuário

4.1 Encontrar máximos e mínimos de funções

Este é o cabeçalho da função principal

```
function [y,x] = EncontrarMaxMinZ(N,T,D,L,C,n_i,f,Comando)
```

- y: Valor máximo ou mínimo encontrado.
- x: Vetor que indica onde o máximo ou mínimo ocorreu.
- N: Número inicial de indivíduos.
- T: Tamanho de cada dimensão dos indivíduos.
- D: Dimensão de cada indivíduo, número de variáveis da função.
- L: Matriz que contém os limites de cada variável, ou seja é uma matriz $2 \times$ D em que cada vetor linha contém o limite inferior e superior das variáveis respectivamente.
- C: Constante que indica a chance de ocorrer uma mutação a cada geração.
- n_i: Número máximo de iterações.
- f: Função de n variáveis a ser otimizada. Atenção, esta função deve ser escrita sempre na forma vetorial ainda que trabalhe apenas com uma variável.
- Comando: *String* que indica qual o desejo do operador entre encontrar um máximo global, um mínimo global ou um zero da função no intervalo fornecido. As strings são respectivamente: "max", "min" e "zero".

Devido a essência aleatória do algoritmo, eis algumas instruções de uso:

1. Devido a natureza aleatória do algoritmo, máximos e mínimos locais podem se encaminhar como a resposta do problema. É recomendável executar o algoritmo mais de uma vez para visualizar tal efeito e aumentar os valores de N, C e n_i para aumentar as chances de se obter a resposta ideal.
2. Aumentar o valor de T para obter maior precisão.
3. O Comando de encontrar zeros encontrará um zero no intervalo dado, mas isto não significa que não há outros valores de x dentro do intervalo que sirva de solução, para encontra-los todos a técnica seria executar o código várias vezes e coletar os valores diferentes alcançados. Para a busca por zeros é melhor usar funções de uma variável somente.

Exemplo de uso:

```
N = 25;
T = 12;
D = 3;
L = [-1,1;-1,1;-1,1];
C = 0.4;
n_i = 1000;
f = @(x) 1 -x(1)^2 - x(2)^2 - x(3)^2;
Comando = 'max';
```

4.2 Problema do Caixeiro Viajante

Este é o cabeçalho da função principal

```
function ProblemaDoCaixeiroViajante(Ncidades,Ncrmossomos,MaxIteracoes)
```

- $Ncidades$: número de cidades que devem ser percorridas.
- $Ncromossomos$: número de cromossomos de uma população.
- $MaxIteracoes$: número máximo de iterações.

Devido a essência aleatória do algoritmo, eis algumas instruções de uso:

1. Devido a natureza aleatória do algoritmo, mínimos locais podem se encaminhar como a resposta do problema. É recomendável executar o algoritmo mais de uma vez para visualizar tal efeito e aumentar os valores de $MaxIteracoes$ para se acrescentar às chances de se obter a resposta ideal.

5 Resultados e Discussões

Aqui serão apresentados análises do uso de algoritmo genético para as aplicações propostas.

5.1 Máximos e Mínimos de funções

Para efeito análise, vamos considerar o caso simples de uma variável. Seja:

$$f(x) = 4\cos(5x) - \sin(5x) + \cos(3x) - 3\sin(3x) + 0.25x$$

para

$$0 < x < 9$$

Deseja-se encontrar o máximo desta função no intervalo fornecido. Para orientação, vamos "trapacear´´ e plotar o gráfico da função:

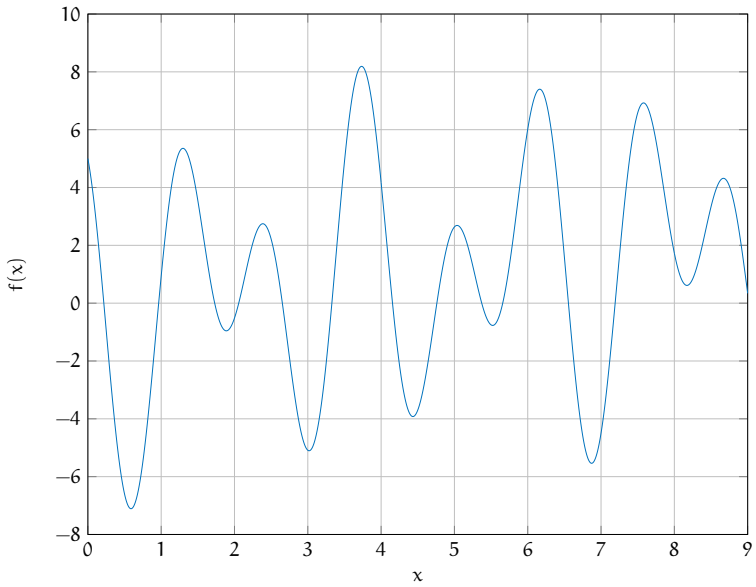

Figura 1: Gráfico da função $f(x) = 4\cos(5x) - \sin(5x) + \cos(3x) - 3\sin(3x) + 0.25x$

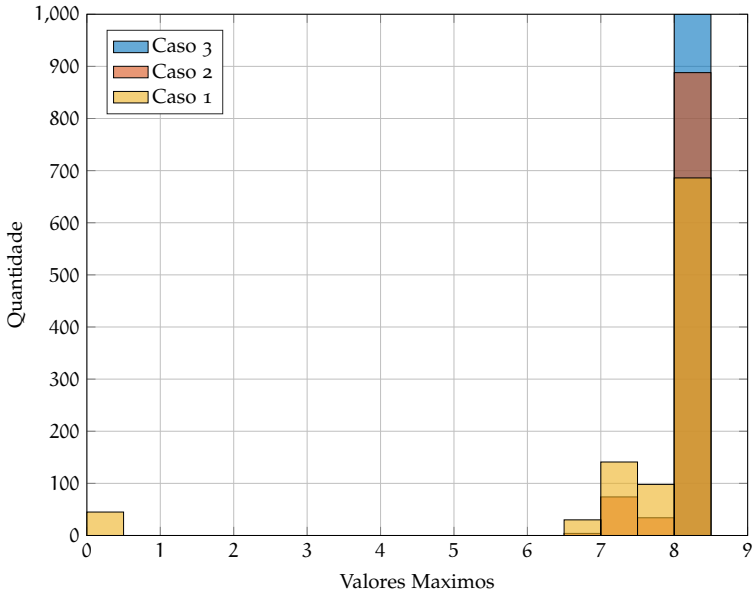

Figura 2: Histograma de mil buscas por máximos

Como se observa, a função tem vários máximos locais mas o seu **máximo global** ocorre para x entre 3.5 e 4.0 e este é claramente maior do que 8. A resposta encontrada pelas execuções do algoritmo foi 8.1891 para $x = 3.7340$.

Foram analisados 3 Casos com os seguintes parâmetros:

1. $N = 20$ e $C = 0.05$
2. $N = 20$ e $C = 0.50$
3. $N = 200$ e $C = 0.50$

A Figura 2 é um histograma que ilustra os valores máximos encontrados em mil execuções do algoritmo para os parâmetros de cada Caso e a Tabela 1 mostra o tempo médio de execução.

Tabela 1: Tabela de tempos médios

Caso	Tempo médio(s)
1	0.0530
2	0.0514
3	0.3198

Os pontos a serem discutidos são:

1. Máximos Locais.
2. O parâmetro C.
3. O parâmetro N.
4. Tempo de execução.

5.1.1 Máximos Locais.

O gráfico da Figura 2, mostra que os Casos 1 e 2 encontraram outros valores como máximo que não estão na faixa de 8 à 8.5. Devido a natureza aleatória do algoritmo, isto é uma possibilidade se os parâmetros não forem adequados. Percebe-se, porém, que o Caso 2, em suas mil buscas, encontrou a resposta correta mais vezes que o Caso 1.

5.1.2 O parâmetro C.

O aumento do valor do parâmetro C incitou maior número de respostas corretas na barra do Caso 2 como mostra a Figura 2, evidenciando a sua importância em livrar a população de uma estagnação local.

5.1.3 O parâmetro N.

O aumento de C seguido do acréscimo em N trouxe a absoluta certeza no resultado como mostra a barra do Caso 3 na Figura 2, as mil buscas trouxeram o valor correto de 8.1891.

5.1.4 Tempo de execução.

A Tabela 1 mostra que o tempo de execução para o Caso 3 foi cerca de 6 vezes maior, o aumento da certeza no resultado exigiu um maior tempo de execução.

5.2 Zeros de funções

Para encontrar zeros de funções um pequeno truque foi empregado, se $f(x)$ é a função a qual se quer encontrar as raízes então uma forma de encontra-la é encontrar o valor máximo de $-|f(x)|$.

A Tabela 2 mostra os valores de x encontrados em 10 buscas do algoritmo. Todos os valores são coerentes com o gráfico da Figura 2, exceto os valores nulos. Isto ocorre porque os limites, as bordas, são máximos locais, então eles devem ser sempre tratados como exceção.

Tabela 2: Raízes encontradas de $f(x)$

x
1,7298
0,21749
1,7298
2,0577
0,0000
5,3724
5,3724
4,1554
0,0000
2,6527

5.3 Problema do Caixeiro Viajante

A análise sobre a busca de máximos e mínimos permitiu concluir alguns defeitos e propriedades do algoritmo. Nesta análise será evidenciado o poder de tal no quesito velocidade.
A Figura 3 mostra um exemplo de solução encontrado pelo algoritmo para o caso de 20 cidades e 3000 iterações. A Figura 4, por sua vez, demonstra a eficácia do algoritmo perante a força bruta, para um número não tão grande como 10 cidades o algoritmo é cerca de 3 vezes mais rápido. Vale ressaltar que a partir de 11 cidades, a solução por Força Bruta se torna inviável, pois exige mais memória que o permitido pelo MATLAB.

6 Discussão e Conclusão Final

6.1 Eficácia

Como ferramentas de otimização, Algoritmos Genéticos são uma válida ferramenta para substituir a força bruta. Contudo, devido à natureza aleatória, uma resposta direta nem sempre é a primeira à surgir. Quando não se possui ideia alguma do máximo ou mínimo global, o melhor caminho é executar o algoritmo várias vezes em ordem para comprovar a resposta.

6.2 Paralelo com a natureza

É no mínimo intrigante ver que os resultados oriundos do algoritmo mimicam a seleção natural, que algo sujeito a mudanças completamente aleatórias mas sujeitos à uma seleção não aleatória encaminhe para respostas cada

vez mais adequadas. Até mesmo os pontos de estagnação e máximos locais podem ser observados na natureza, como é o caso por exemplo do Ornitorrinco, animal que herda características de ancestrais à 250 milhões de anos atrás. [3]

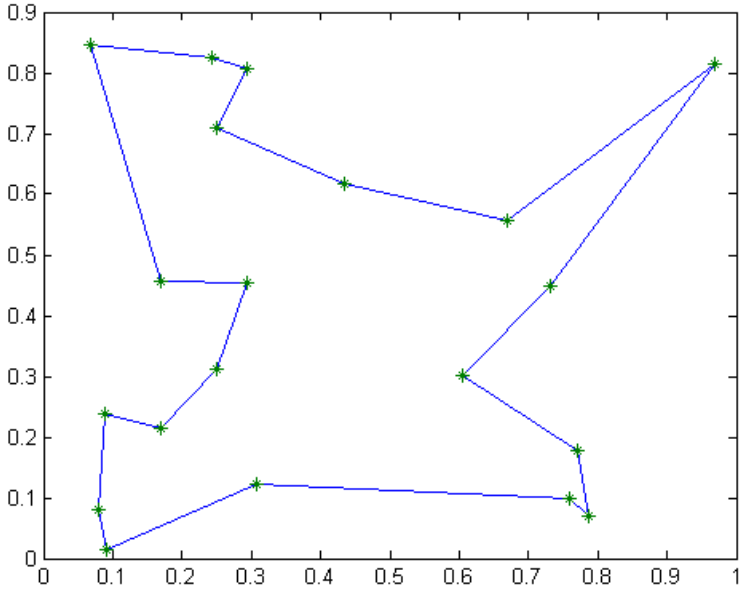

Figura 3: Exemplo de Solução PCV.

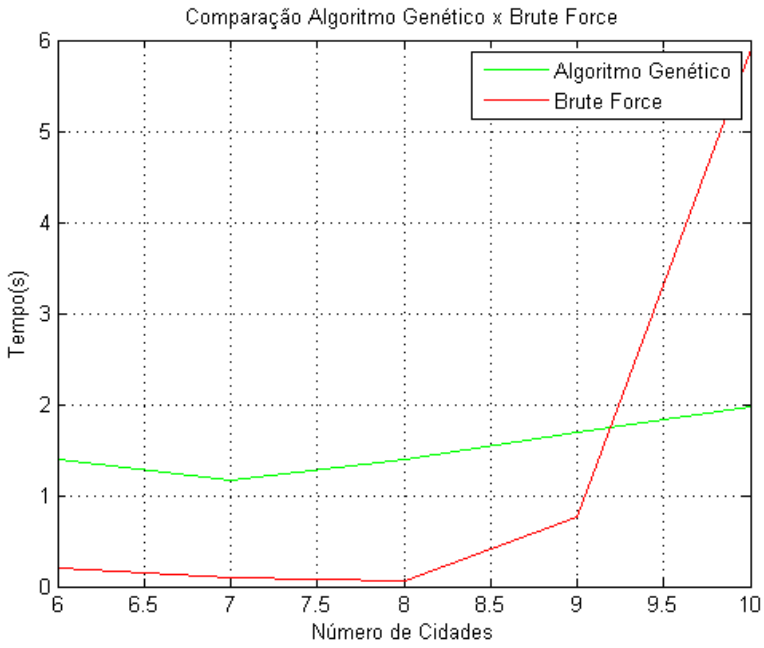

Figura 4: Comparação entre métodos.

REFERÊNCIAS

[1] https://karczmarczuk.users.greyc.fr/TEACH/IAD/GenDoc/carrGenet.pdf

[2] http://www.mat.ufrgs.br/~portosil/caixeiro.html

[3] http://www.bbc.com/earth/story/20150413-can-an-animal-stop-evolving