ANTÔNIO CARLOS DURÃES DA SILVA
JOEL WILL BELMIRO

GitHub: https://github.com/duraes-antonio/IA_trab3_AG

INTELIGÊNCIA ARTIFICIAL – 3º TRABALHO: ALGORÍTIMO GENÉTICO COM SELEÇÃO POR TORNEIO E CÓDIGO GENÉTICO EM FORMATO BINÁRIO

1 EXPLICAÇÃO TEÓRICA DO ALGORITMO

Inspirado em conceitos pertencentes à Teoria da Seleção Natural, registrada em nome de Charles Robert Darwin [1], o algorítimo genético visa tirar proveito de mecanismos semelhantes aos naturais para selecionar, modificar ou criar uma nova solução para um problema a partir de soluções pais.

O algorítimo genético herda da Biologia não só os mecanismos para encontrar a melhor solução mas também a modelagem utilizada para definir a estrutura de uma solução, ou melhor, de um indivíduo. Cada valor que representa uma solução pode ser modelado como um indivíduo, contendo seu código genético e sua aptidão.

É possível traçar uma analogia, em que uma fórmula matemática ou objetivo representa esse ambiente responsável por definir quem é mais adequado. Dessa forma, o código genético (simboliza um parâmetro) do indivíduo aliado ao ambiente (fórmula ou processamento que recebe a entrada) determina o quão apto (valor fitness) cada indivíduo é, sendo possível, selecionar tal indivíduo, reproduzi-lo e impulsionar pequenas alterações em sua genética.

O algoritmo favorece a descoberta de bons resultados para, principalmente, problemas em que sua solução final não é conhecida ou que não há uma solução ótima alcançável [2], sendo satisfatórias, soluções que oscilem entre uma faixa de aceitabilidade, graças à sua aleatoriedade, seleção e variação de soluções promissoras.

2 PROBLEMA PROPOSTO

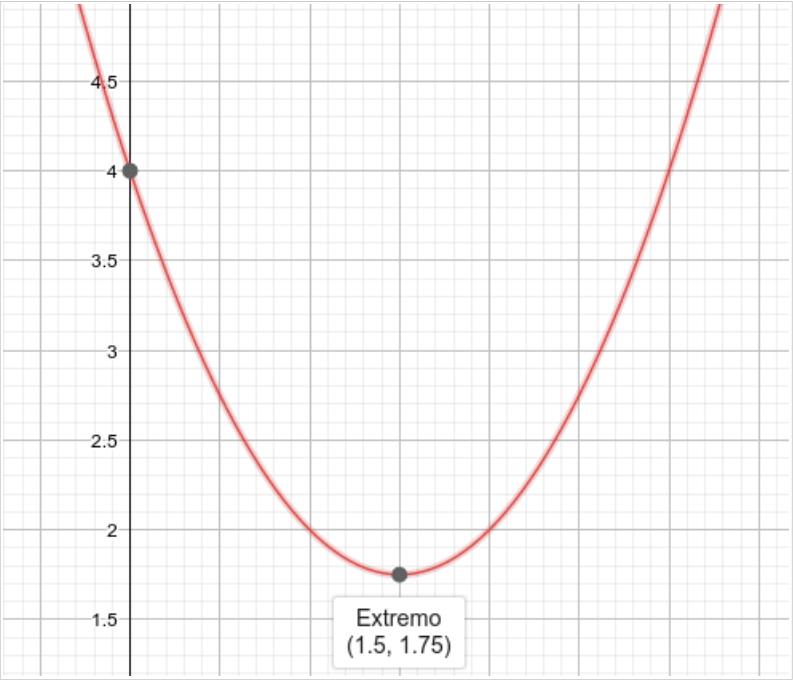
O algorítimo teve como tarefa encontrar o valor para o parâmetro X que minimizasse o resultado da função abaixo:

Fórmula 1: Função a ser minimizada

$$f(x) = x^2 - 3x + 4$$

Após plotar o gráfico 2D da função acima na ferramenta Geogebra, fica claro que seu ponto de mínimo é de 1.75, e é atingido para X com valor de 1.5.

Gráfico 1: Plotagem da fórmula 1



3 IMPLEMENTAÇÃO

3.1 INDIVÍDUO

Modelagem: Cada indivíduo foi modelado como uma classe contendo seu número de bits (comprimento de seu cromossomo), a cadeia de bit em si, um valor máximo e um valor mínimo de aptidão.

```
class Individuo(object):
    """Representa um indivíduo biológico com código genético e valor de aptidão."""
    n_bits = 10 # Comprimento do cromossomo
    dMax = 10   # Valor máximo dentro do domínio da função
    dMin = -10  # Valor mínimo dentro do domínio da função
```

Instanciação: Para instanciar um indivíduo é necessário passar o tamanho de sua cadeia genética. Também é possível passar uma cadeia genética já existente, para 'clonar' um indivíduo.

```
def __init__(self, n_bits: int, bits: str = None):
    """Instancia um indivíduo novo ou a partir de um código genético existente."""
    self.n_bits = n_bits

    # Senão informar os bits, eles são gerados
    self.bits = bits if bits else self.generate_bits()
```

Geração de código genético: Se a cadeia de bits estiver vazia, a função abaixo gera uma cadeia randomicamente para inicializar o código genético do indivíduo.

```
def __generate_bits(self) -> str:
    """
    :return: String de tamanho Individuo.n_bits com cada caractere podendo ser 0 ou 1
    """
    return "".join([str(random.randint(0, 1)) for i in range(self.n_bits)])
```

3.2 POPULAÇÃO

Modelagem: A população ficou responsável por armazenar informações como o número de indivíduos, a taxa de mutação, taxa de crossover, além de armazenar o melhor (indivíduo elitizado) a cada geração.

```
class Populacao(object):
    """Representa um coletivo. Contém operações de seleção, crossover e mutação."""
    elite = None     # Não há elite até a população ser preenchida
    n_ind = 0 # Só haverá indivíduos quando a população for preenchida

    def __init__(self, tx_mutacao: int, tx_cross: int, n_individ: int, n_bits: int, interv_min: int = None,
                 interv_max: int = None):
        # Atualize as propriedades com os valores recebidos
        self.n_ind = n_individ
        self.__taxa_mut = tx_mutacao
        self.__taxa_cross = tx_cross
        self.__n_bits = n_bits

        # Gera os indivíduos da população
        self.individuos: [Individuo] = [Individuo(n_bits=n_bits) for i in range(self.n_ind)]
        self.elite = Individuo(self.__n_bits, self.get_best_or_worst().bits)
```

Elitização: O processo de elitização garante que entre os indivíduos de cada geração, haverá o indivíduo com o código genético mais apto de todas gerações até então.

```
    def __apply_elite(self):

        # Obtenha o mais apto indivíduo
        temp_best = self.get_best_or_worst()

        # Se o indivíduo acima for mais apto que o da elite, atualize a elite
        if temp_best.fitness <= self.elite.fitness:
            self.elite = Individuo(self.__n_bits, temp_best.bits)

        # Se o pior individuo for menos apto que o da elite
        else:
            worst = self.get_best_or_worst(best=False)

            # Substitua o pior pela elite
            if worst.fitness > self.elite.fitness:
                idx = self.individuos.index(worst)
                self.individuos[idx] = Individuo(self.__n_bits, self.elite.bits)
```

Seleção: O processo de seleção utiliza a técnica de torneio, ou seja, N indivíduos (neste caso, n = 2) são sorteados e o que for mais apto permanece na população. O processo ocorre K (número de indivíduos na população) vezes.

```
def select(self):
    """Seleciona os melhores indivíduos usando método de torneio com n = 2"""

    inds_selected = []

    # Para i, de 0 até o número de indivíduos
    for i in range(self.n_ind):

        # Escolha aleatoriamente 2  indivíduos
        ind1 = choice(self.individuos)
        ind2 = choice(self.individuos)

        # O Individuo selecionado será o de menor fitness
        inds_selected.append(ind1 if ind1.fitness <= ind2.fitness else ind2)

    # Atualiza a população para os indivíduos que foram selecionados
    self.individuos = inds_selected

    # Aplica a elitização
    self.__apply_elite()
```

Crossover: O processo de cruzamento é responsável por gerar novos indivíduos a partir de dois indivíduos pais. Além da taxa de crossover, o ponto onde o DNA dos pais é cortado define significativamente a aptidão do indivíduo filho.

```
def make_crossover(self):
    """Faz crossover entre indivíduos pelo método de 1 corte"""
    children: [Individuo] = []

    while len(children) <= self.n_ind:
        # Sorteia a taxa de crossover de 0% a 100%
        tax = randint(0, 100)

        # Escolhe aleatoriamente 2 indivíduos
        ind1: Individuo = choice(self.individuos)
        ind2: Individuo = choice(self.individuos)

        # Se a taxa estiver no valor aceitável
        if tax <= self.__taxa_cross:

            # Sorteia a posição de corte
            cut_pos = randint(1, Individuo.n_bits - 2)

            # Gera os bits
            bits1 = ind1.bits[:cut_pos] + ind2.bits[cut_pos:]
            bits2 = ind2.bits[:cut_pos] + ind1.bits[cut_pos:]

            # Gera os filhos
            children.append(Individuo(self.__n_bits, bits1))
            children.append(Individuo(self.__n_bits, bits2))

        # Se a taxa não ficou dentro do aceitável
        else:

            # Adiciona os pais como filhos
            children.append(ind1)
            children.append(ind2)

    # Muda os indivíduos para os filhos gerados
    self.individuos = children
    self.__apply_elite()
```

4 EXEMPLO DE USO

4.1 EXEMPLO DE CHAMADA E ENTRADAS

O algoritmo foi estruturado sob uma aplicação de linha de comando (CLI), portanto, os parâmetros e taxas envolvidas são livres para entrada do usuário. Abaixo está um exemplo de chamada num terminal Linux com 4 indivíduos na população, 1% de taxa de mutação, e 60% de taxa de crossover respectivamente:

```
$ python3 main.py -i 4 -m 1 -c 60
```

Em caso de dúvidas sobre cada parâmetro, basta chamar a aplicação passando o parâmetro '-h':

```
$ python3 main.py -h
```

Observação: Para sistemas Windows, a chamada é a mesma, porém em vez de utilizar “python3” para chamar a linguagem, usa-se apenas “python”.

Explicação sobre parâmetros de entrada:

- -i / --individuo: Número de indivíduos;
- -m / --mutacao: Porcentagem de chance de ocorrer mutação por bit;
- -c / --crossover: Porcentagem de chance de ocorrer crossover entre 2 indivíduos;

4.2 SAÍDA

A saída após a execução do programa é um arquivo .CSV para cada iteração e configuração. Por exemplo, para 5 gerações, 10 execuções, serão gerados 10 arquivos, um para cada execução.

O nome do arquivo segue o padrão:

{número_indivíduos}i_{número_gerações}g_{número_execução_atual}exec.csv

Exemplos: 4i_5g_1exec.csv, 4i_5g_2exec.csv, 4i_5g_3exec.csv, ...

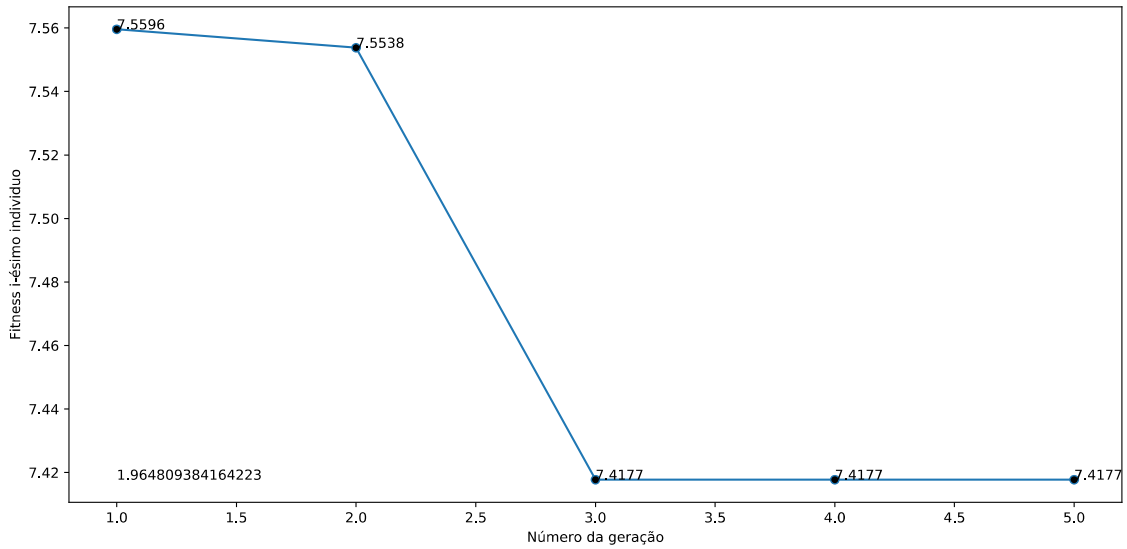
Cada linha do arquivo contém o número da geração, os bits do indivíduo mais apto, seu valor de X, o valor normalizado de X e valor obtido a partir da função fitness:

```
1;Bits = 1001010110;X = 598;X_normalizado = 1.691104594330401;Fitness = 1.7865209659741
```

5 RESULTADOS E OBSERVAÇÕES

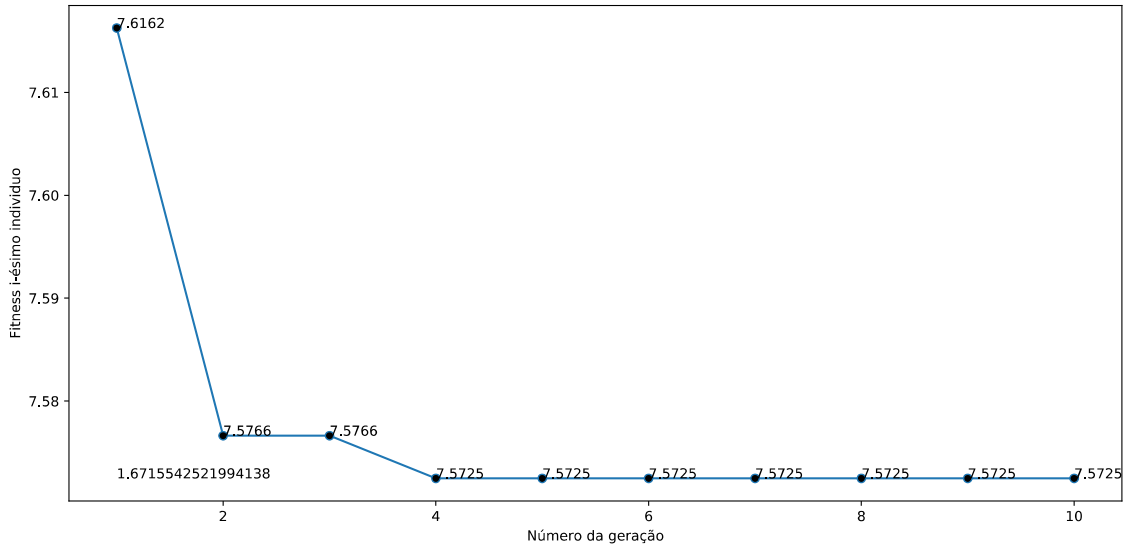
Embora a natureza do algoritmo seja randômica, isto é, para cada execução pode resultar em alto ou baixo sucesso; para fins de simples comparação, abaixo no gráfico 2 é possível ver o resultado de uma chamada com 10 execuções e 5 gerações.

Gráfico 2: Resultado com 4 indivíduos, 1% de chance de mutação, 60% de crossover, 5 gerações e 10 execuções.



Já no gráfico 3 é possível ver o resultado de uma chamada com 10 execuções e 10 gerações.

Gráfico 3: Resultado com 4 indivíduos, 1% de chance de mutação, 60% de crossover, 10 gerações e 10 execuções.



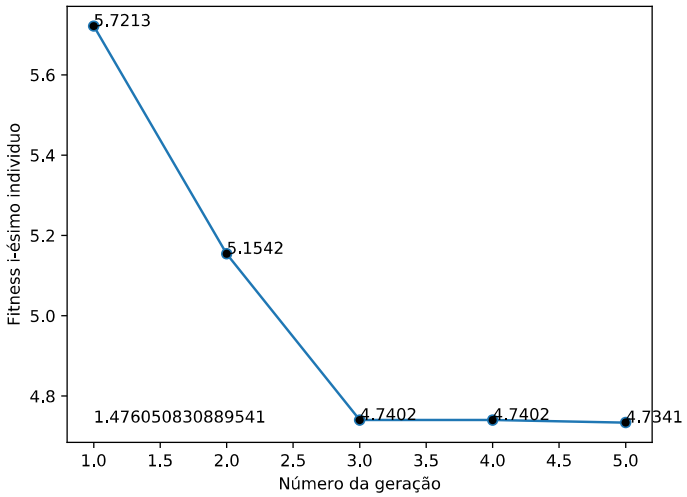
Nota-se que o aumento do número de gerações com a mesma quantidade de indivíduos e taxa de mutação não resultou em melhorias significativas.

5.1 RELAÇÃO: TAMANHO DA POPULAÇÃO E PRECISÃO

O gráfico 4 indica que além começar mais próximo do fitness final (1.75), a melhoria absoluta vista entre a primeira geração e a última é de 0.9872 (5.7213 – 4.7341).

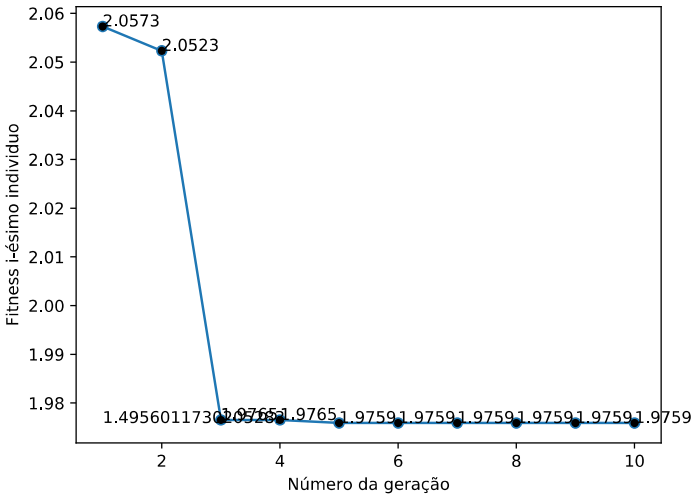
Tal êxito pode ter sido alcançado devido ao aumento de chance de um dos indivíduos ser gerado com código genético próximo ao objetivo. Com 8 indivíduos e o fator de aleatoriamente, dobra-se a chance de a cada geração um indivíduo nascer com genética propensa ao valor adequado para x, que é de 1.5.

Gráfico 4: Resultado com 8 indivíduos, 1% de chance de mutação, 60% de crossover, 5 gerações e 10 execuções.



O gráfico 5 ratifica a hipótese do indivíduo nascer próximo ao resultado final (1.75), com a primeira geração já obtendo indivíduos próximo ao fitness de valor 2.

Gráfico 5: Resultado com 8 indivíduos, 1% de chance de mutação, 60% de crossover, 10 gerações e 10 execuções.



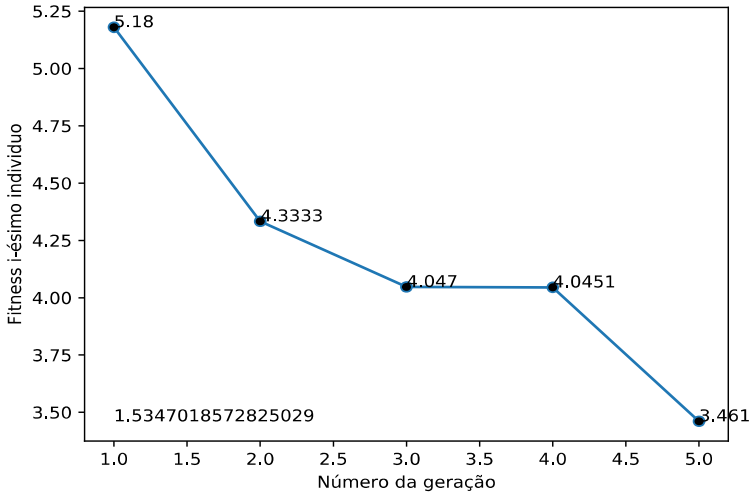
5.2 RELAÇÃO: TAXA DE MUTAÇÃO E PRECISÃO

A alteração da taxa de mutação é o segundo fator que acelera a convergência do algoritmo. Tal acréscimo implica no aumento da variabilidade genética dos indivíduos, o que possibilita que indivíduos menos aptos sofram modificações que te tornem mais assertivos às condições, ou na prática, com a cadeia de bits resultando em fitness mais próxima de 1.75.

A mutação permite também que indivíduos nascidos aptos tornem-se pouco promissores, o que ocasionaria picos no gráfico, algo que só não ocorre porque o processo de elitização garante sempre haverá o melhor indivíduo na população, e este só será substituído por um mais apto que ele.

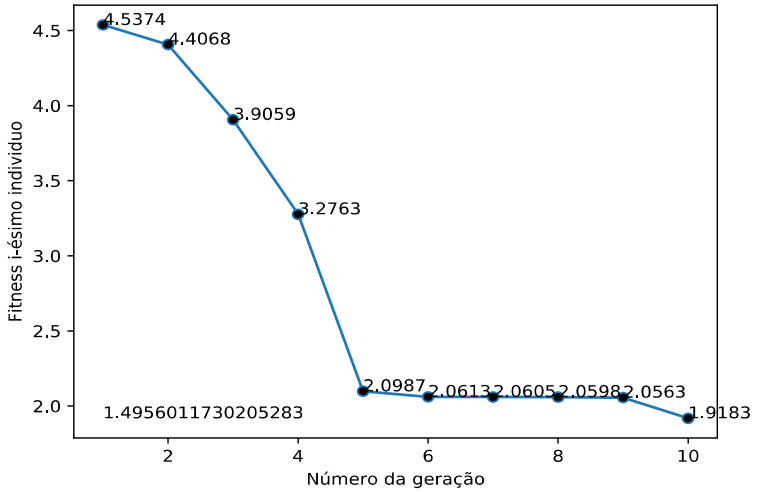
No gráfico 6, com uso de 5% de mutação, é possível perceber que na primeira geração há indivíduos mais promissores que na chamada que utilizou 1% de mutação. Também é possível notar a variação (de 1.719) entre a primeira geração e a última.

Gráfico 6: Resultado com 4 indivíduos, 5% de chance de mutação, 60% de crossover, 5 gerações e 10 execuções.



No gráfico 7, com 10 gerações, da primeira até a quinta geração, a variação é significativa e constante, a partir de então, a variação é reduzida. O salto da primeira até a última geração é de 2.619.

Gráfico 7: Resultado com 4 indivíduos, 5% de chance de mutação, 60% de crossover, 10 gerações e 10 execuções.



5.3 RELAÇÃO: CADEIA GENÉTICA (NÚMERO DE BITS) E PRECISÃO

O comprimento da cadeia genética e a convergência do algoritmo é uma das relações menos perceptíveis se trabalhada só, sem auxílio do aumento da taxa de mutação. Com o domínio de X variando entre 10 e -10, e auxílio da função de normalização é possível verificar que o aumento da quantidade de bits implica no aumento da precisão e faixa de valores possíveis para X.

Abaixo, na tabela 1, é possível conferir as possibilidades para um cromossomo com tamanho de 2 bits. Com 4 valores é fácil atingir o número (~3.33333) mais próximo de 1.5, só é necessário acertar 1 em 4 valores, isto é 25% chance. Contudo, pelo fato de sua precisão ser baixa, esta precisão será refletida no valor final encontrado, ou seja, o valor mais próximo, ~3.3333, apresenta alta variação se comparado ao 1.5.

Tabela 1: Resultados para X (binário, decimal e normalizado) com 2 bits

Binário	Decimal	Normalizado
0000	0	-10
0001	1	-3,3333
0010	2	3,3333
0011	3	10

Com 4 bits, há 16 possibilidades de valores para X, valores estes que oscilam entre 0 e 15. A chance de encontrar o valor mais próximo de 1.5 reduz de 25% para 6.25% (1/16), em contrapartida, além do valor mais próximo ser mais preciso, há um conjunto muito maior de valores intermediários, ou seja, a chance de encontrar o valor mais próximo do X desejado pode ser mais baixa, mas de encontrar valores próximos ao desejado aumentou.

Tabela 2: Resultados para X (binário, decimal e normalizado) com 4 bits

Binário	Decimal	Normalizado
0000	0	-10
0001	1	-8,6667
0010	2	-7,3333
0011	3	-6
0100	4	-4,6667
0101	5	-3,3333
0110	6	-2
0111	7	-0,6667
1000	8	0,6667
1001	9	2
1010	10	3,3333
1011	11	4,6667
1100	12	6
1101	13	7,3333
1110	14	8,6667
1111	15	10

O aumento excessivo do tamanho da cadeia de bits é um erro, pois a precisão aumenta de tal forma que a diferença entre um valor e seu seguinte é pouco sensível ao domínio da função. Para uma cadeia de 20 bits, por exemplo, a diferença entre um valor X_i e seu X_{i+1} é de 1.9073504518019035e-05.

6 REFERÊNCIAS E OUTROS MATERIAIS BASE

[1]. Página 43. Acesso em 25/05/2019. Disponível em: http://revista.pgsskroton.com.br/index.php/rcext/article/view/2394/2298

[2]. Página 2. Acesso em 25/05/2019. Disponível em: http://www.fsma.edu.br/si/edicao3/aplicacoes_de_alg_geneticos.pdf