

**Murilo Cesar Osorio**

# TP1: Algoritmos Genéticos

Trabalho apresentado ao Professor  
João Batista Mendes da disciplina  
Computação Evolutiva do 6º período  
do curso de Engenharia de Sistemas.

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Objetivos</b>	<b>3</b>
<b>3</b>	<b>Materiais e Métodos</b>	<b>3</b>
3.1	Classe Pop . . . . .	3
3.2	Classe GA . . . . .	4
<b>4</b>	<b>Resultados e Discussões</b>	<b>4</b>
4.1	Teste 1: Avaliação do efeito do tipo de cruzamento . . . . .	4
4.2	Teste 2: Avaliação do efeito do tipo de seleção . . . . .	5
4.3	Teste 3: Avaliação do efeito do tipo de mutação . . . . .	5
4.4	Teste 4: Avaliação do efeito da probabilidade de cruzamento . . . . .	5
4.5	Teste 5: Avaliação do efeito da probabilidade de mutação . . . . .	6
4.6	Avaliação Final . . . . .	6
<b>5</b>	<b>Conclusão</b>	<b>7</b>

# 1 Introdução

O trabalho consiste na utilização de algoritmos genéticos e seus operadores para resolução de um problema do tipo caixa preta cuja função de saída é dada pela Equação 1. A representação do problema é binária e as soluções candidatas possuem 36 bits,  $b_1$  corresponde ao primeiro, ao passo que  $b_{36}$  corresponde ao último bit da *string*.

$$\begin{aligned} y_{saida} = & 9 + b_2b_5 - b_{23}b_{14} + b_{24}b_4 - b_{21}b_{10} + b_{36}b_{15} - b_{11}b_{26} + b_{16}b_{17} \\ & + b_3b_{33} + b_{28}b_{19} + b_{12}b_{34} - b_{31}b_{32} - b_{22}b_{25} + b_{35}b_{27} - b_{29}b_7 \\ & + b_8b_{13} - b_6b_9 + b_{18}b_{20} - b_1b_{30} + b_{23}b_4 + b_{21}b_{15} + b_{26}b_{16} \\ & + b_{31}b_{12} + b_{25}b_{19} + b_7b_8 + b_9b_{18} + b_1b_{33} \end{aligned} \quad (1)$$

É possível perceber, levando a zero os termos que são subtraídos e a um os que são somados, que o valor máximo da Equação 1 é **27**. Portanto, vinte e sete corresponde ao *fitness* ótimo para este problema.

## 2 Objetivos

Os objetivos deste trabalho são de aprender os conceitos que permeiam o tópico de estudos de Algoritmos Genéticos através da implementação de seus operadores e algumas variantes de cada um deles. Além disso, por ter um estudo estatístico envolvido, o trabalho permite que o aluno possa aprender via gráficos as nuances do dimensionamento de um algoritmo genético, no que diz respeito ao seu comportamento de acordo com a escolha dos operadores, população inicial, taxas de mutação e cruzamento, e diversos outros parâmetros ajustáveis.

## 3 Materiais e Métodos

O trabalho foi implementado utilizando a linguagem **Python 2.7** com auxílio da bibliotecas **Numpy** e **Matplotlib**. A modelagem foi feita utilizando conceitos de orientação a objetos por tornar mais fácil a compreensão do código e deixar mais claro o encapsulamento dos métodos e atributos.

Foram criadas duas classes; a classe **Pop** modela uma população de tamanho  $T$ , dimensão  $D$  e uma função de *fitness*, os métodos dessa classe são os operadores de mutação, seleção, cruzamento e substituição; a segunda classe (**GA**) é responsável por processar o dicionário de configurações que o usuário pode informar e realizar os testes. O teste estatístico de Wilcoxon foi calculado utilizando a biblioteca **stats** do pacote **scipy**.

### 3.1 Classe Pop

Esta classe generaliza uma população com atributos de tamanho ( $T$ ) e dimensão ( $D$ ). A população é representada por uma matriz ( $M$ ) *np.array* com  $N$  linhas (indivíduos) e  $D$  colunas; o *fitness* é definido pelo usuário como uma função que recebe uma linha dessa matriz e retorna um valor que determina a qualidade da solução.

Os métodos representados por esta classe são:

1. **eval**: recebe a matriz  $M$  como entrada e retorna um vetor  $F^{1 \times N}$  com os valores de *fitness* de cada solução.
2. **selection**: realiza a operação de seleção na população  $M$ , usando o algoritmo de torneio binário ou roleta (a ser definido pelo usuário) e retorna uma nova população  $S$ .
3. **crossover**: realiza a operação de cruzamento na população  $S$ , usando o algoritmo de cruzamento uniforme ou com um ponto de corte. Retorna uma nova população  $C$ .

4. **mutation**: faz mutações na população  $C$  com as metodologias bit-a-bit e uniforme. Retorna uma nova população  $E$ .
5. **substitution**: faz a substituição da população  $E$  na população  $M$  inicial utilizando elitismo.

### 3.2 Classe GA

Esta classe recebe as configurações do algoritmo fornecidas pelo usuário e realiza os testes, também determinados pelo usuário. Seu valor de inicialização é um dicionário com as seguintes chaves:

- **popSize**: <inteiro> tamanho da população;
- **popDim**: <inteiro> dimensão da população;
- **representation**: <str> tipo de representação ['binary'];
- **fitnessEval**: <object> uma função python para calculo de *fitness*;
- **crossRate**: <float> taxa de cruzamento [0,1];
- **crossType**: <str> tipo de cruzamento ['uniform', '1cp'];
- **selectionType**: <str> tipo de seleção ['roulette', 'tournament'];
- **mutationRate**: <float> taxa de mutação [0,1];
- **mutationType**: <str> tipo de mutação ['uniform', '1bit'];
- **maxEpochs**: <int> numero máximo de épocas.

Para realizar os testes, a função **test** implementa um GA geral e recebe como parâmetros a quantidade de testes a serem realizados com as configurações listadas anteriormente e o nome do arquivo que será salvo com os resultados. Cada um dos testes é inicializado com uma população aleatória; ao final do teste, o maior *fitness* da população é inserido num vetor que será retornado ao final de todos os testes.

## 4 Resultados e Discussões

### 4.1 Teste 1: Avaliação do efeito do tipo de cruzamento

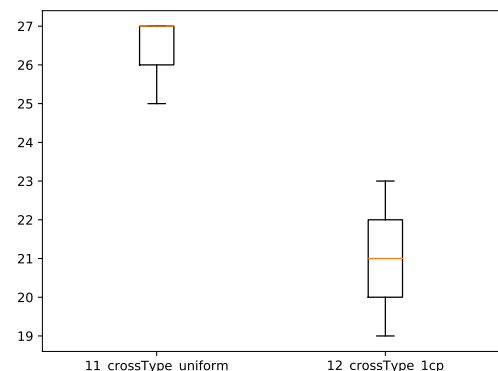
Os parâmetros para a realização deste teste foram: seleção por roleta, mutação bit a bit, cruzamentos uniforme e com um ponto de corte, taxa de cruzamento igual a 0.8, taxa de mutação igual a 0.025, número de indivíduos igual a 30 e limitada a 50 gerações. Pelos resultados abaixo, podemos perceber que a melhor configuração para o tipo de cruzamento é o uniforme. É visível, pelo *boxplot*, que o primeiro método de cruzamento teve resultado melhor que o segundo. O teste de Wilcoxon resultou em **2.52e-34**, ou seja, os resultados são estatisticamente diferentes.

#### Uniforme:

- Sucessos: **54**
- Maior *fitness*: **27**
- Menor *fitness*: **25**
- $\mu_{fitness}$ : **26.52**
- $\sigma_{fitness}$ : **0.61**

#### 1pc:

- Sucessos: **0**
- Maior *fitness*: **23**
- Menor *fitness*: **19**
- $\mu_{fitness}$ : **21.12**
- $\sigma_{fitness}$ : **1**



## 4.2 Teste 2: Avaliação do efeito do tipo de seleção

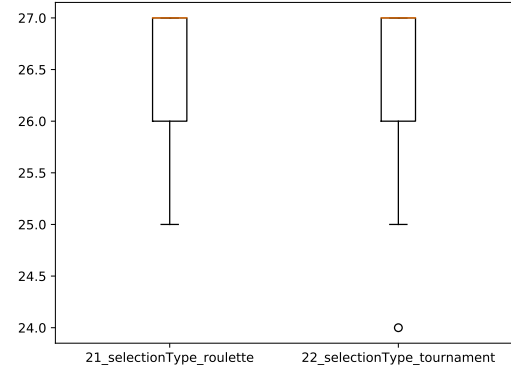
O teste de Wilcoxon para este resultado foi **1.34e-6**, indicando que a diferença entre os resultados foi pequena. De fato, a utilização do método da roleta ou do torneio binário não influenciaram muito no resultado, no entanto, o método da roleta foi ligeiramente melhor nos testes.

### Roleta:

- Sucessos: **58**
- Maior *fitness*: **27**
- Menor *fitness*: **25**
- $\mu_{fitness}$ : **26.52**
- $\sigma_{fitness}$ : **0.61**

### Torneio:

- Sucessos: **56**
- Maior *fitness*: **27**
- Menor *fitness*: **24**
- $\mu_{fitness}$ : **26.5**
- $\sigma_{fitness}$ : **0.63**



## 4.3 Teste 3: Avaliação do efeito do tipo de mutação

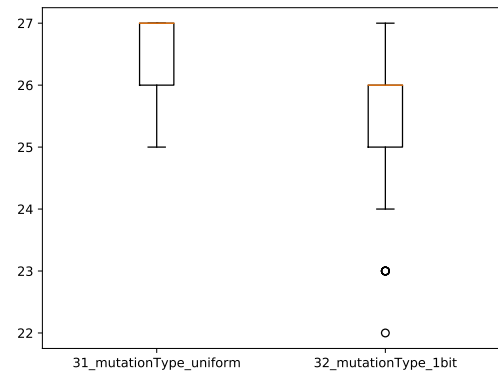
O teste de Wilcoxon para este resultado foi **0.06**, indicando que a diferença entre os resultados foi pequena. De fato, a utilização do método da mutação uniforme ou bit-a-bit não influenciaram muito no resultado, no entanto, o método uniforme foi melhor nos testes no que diz respeito às piores e melhores soluções e na quantidade de sucessos. O *boxplot* mostra os valores fora da curva, chamados *outliers*.

### Uniforme:

- Sucessos: **58**
- Maior *fitness*: **27**
- Menor *fitness*: **25**
- $\mu_{fitness}$ : **26.52**
- $\sigma_{fitness}$ : **0.61**

### Bit-a-bit:

- Sucessos: **19**
- Maior *fitness*: **27**
- Menor *fitness*: **22**
- $\mu_{fitness}$ : **25.52**
- $\sigma_{fitness}$ : **1.14**



## 4.4 Teste 4: Avaliação do efeito da probabilidade de cruzamento

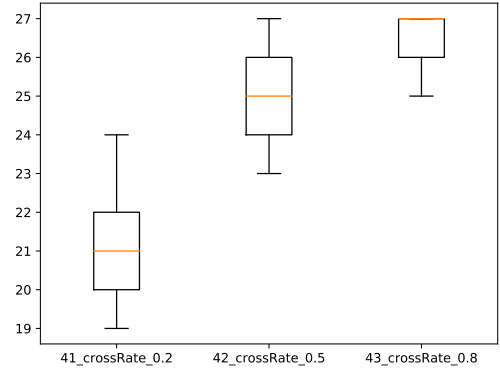
Os valores para  $P_C = 0.8$  já foram mostrados anteriormente. Pode-se perceber que  $P_C = 0.8$  retorna os melhores resultados. Os valores do teste de Wilcoxon para as amostras ( $P_C = 0.2$ ,  $P_C = 0.5$ ), ( $P_C = 0.2$ ,  $P_C = 0.8$ ) e ( $P_C = 0.5$ ,  $P_C = 0.8$ ) são, respectivamente, **2.52e-34**, **2.52e-34** e **6.79e-31**; isto indica que os resultados são estatisticamente diferentes.

$P_C = 0.2$

- Sucessos: **0**
- Maior *fitness*: **24**
- Menor *fitness*: **19**
- $\mu_{fitness}$ : **20.95**
- $\sigma_{fitness}$ : **1.24**

$P_C = 0.5$

- Sucessos: **7**
- Maior *fitness*: **27**
- Menor *fitness*: **23**
- $\mu_{fitness}$ : **24.96**
- $\sigma_{fitness}$ : **1.09**

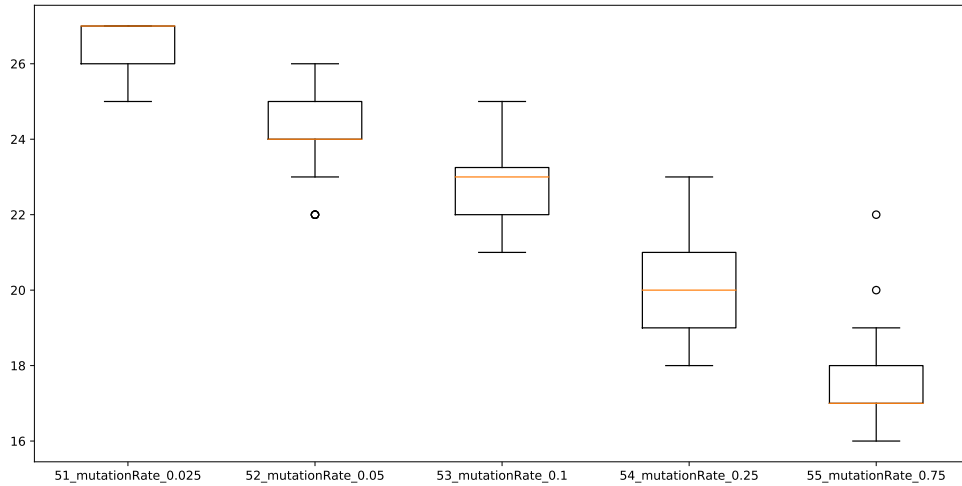


#### 4.5 Teste 5: Avaliação do efeito da probabilidade de mutação

Pode-se perceber, com as informações obtidas, que a melhor configuração continua sendo a configuração anterior com  $P_M = 0.025$ . Isto é visível também nos boxplots.

$P_M$	0.05	0.1	0.5	0.75
<b>Sucessos</b>	0	0	0	0
<b>Maior <i>fitness</i></b>	26	25	23	22
<b>Menor <i>fitness</i></b>	22	21	18	16
<b><math>\mu_{fitness}</math></b>	24.14	22.76	20.09	17.61
<b><math>\sigma_{fitness}</math></b>	0.92	0.96	1.06	0.9

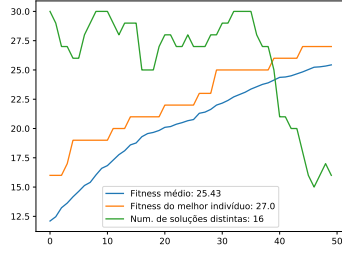
Teste 1	Teste 2	Wilcoxon
$P_M = 0.025$	$P_M = 0.05$	2.72e-34
$P_M = 0.025$	$P_M = 0.1$	2.52e-34
$P_M = 0.025$	$P_M = 0.25$	2.52e-34
$P_M = 0.025$	$P_M = 0.75$	2.52e-34
$P_M = 0.05$	$P_M = 0.1$	6.75e-29
$P_M = 0.05$	$P_M = 0.25$	2.52e-34
$P_M = 0.05$	$P_M = 0.75$	2.52e-34
$P_M = 0.1$	$P_M = 0.25$	2.52e-34
$P_M = 0.1$	$P_M = 0.75$	2.52e-34
$P_M = 0.25$	$P_M = 0.75$	2.52e-34



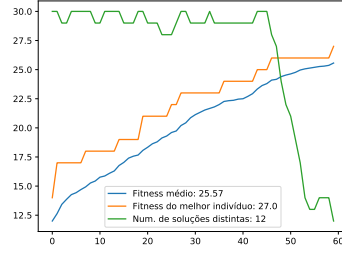
#### 4.6 Avaliação Final

Utilizando a configuração “ótima” encontrada através dos testes, e variando o número máximo de gerações ( $K$ ) e de indivíduos ( $T$ ), temos alguns resultados interessantes. É possível perceber que, eventualmente, o *fitness* médio da população atinge o valor de *fitness* máximo; isto se dá pelo fato de o elitismo manter as melhores soluções da população para a próxima

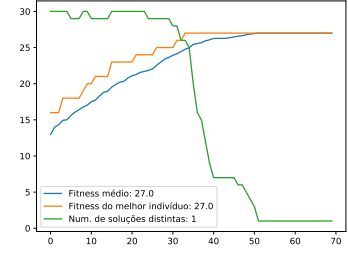
iteração. O problema disso é que, ao final da execução, temos poucas soluções distintas de fato; o algoritmo decidiu por manter soluções iguais!



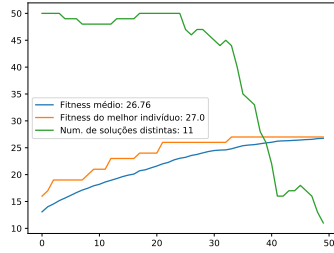
(a)  $(T, K) = (30, 50)$



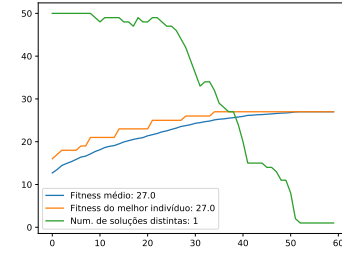
(b)  $(T, K) = (30, 60)$



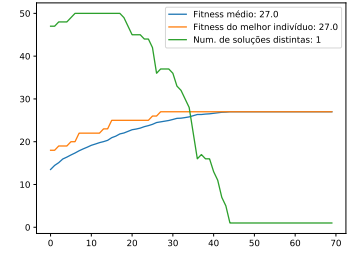
(c)  $(T, K) = (30, 70)$



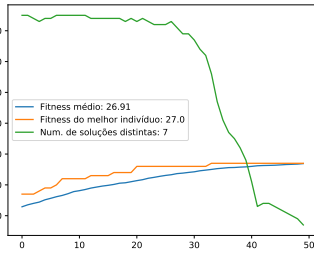
(d)  $(T, K) = (50, 50)$



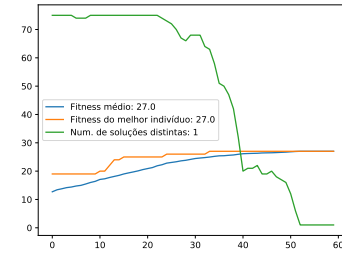
(e)  $(T, K) = (50, 60)$



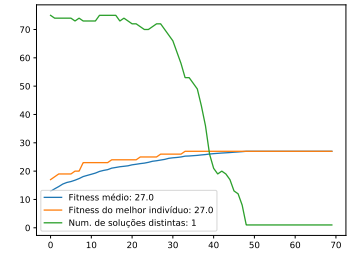
(f)  $(T, K) = (50, 70)$



(g)  $(T, K) = (75, 50)$



(h)  $(T, K) = (75, 60)$



(i)  $(T, K) = (75, 70)$

## 5 Conclusão

O trabalho foi uma forma de aprendizado dos algoritmos genéticos através de sua implementação; para este trabalho foi escolhida a linguagem Python 2, por simples afinidade. Uma vez que os conceitos de lógica de programação e peculiaridades da linguagem específica fazem parte do conhecimento solidificado do aluno, a implementação não deve apresentar grandes dificuldades.

A comparação entre as diversas opções de parâmetros foi feita através dos resultados numéricos e mostrada através de *boxplots*, que conseguem ilustrar a situação geral da população no que diz respeito aos seus quartis. Através dessa análise, pode-se encontrar uma configuração “ótima” para o problema em questão e foi possível perceber que o *fitness* médio da população tende para o *fitness* do melhor indivíduo quando o elitismo é usado. Como mostrados nos gráficos da seção 4.6, a população perde bastante (e de forma rápida) sua heterogeneidade ao eliminar as piores soluções.

## Referências

- [1] Fred L. Drake Jr. The python standard library. <https://docs.python.org/2/library/>. [Acessado em 04/02/2017].
- [2] Travis E. Oliphant. Numpy reference. <https://docs.scipy.org/doc/numpy/reference/>, 2008. [Acessado em 04/02/2017].
- [3] Statistical functions. <https://docs.scipy.org/doc/scipy-0.18.1/reference/stats.html>. [Acessado em 04/02/2017].
- [4] Pyplot tutorial. [http://matplotlib.org/users/pyplot\\_tutorial.html](http://matplotlib.org/users/pyplot_tutorial.html). [Acessado em 04/02/2017].
- [5] Greg von Winckel. Find unique rows in numpy.array. <http://stackoverflow.com/a/22941699>, 2014. [Acessado em 04/02/2017].