
Otimizações de build com GA e ES para Genshin Impact

— Tema: Otimização Combinatória —

Aluno: Victor Gabriel Tenório Oliveira

Definição do Problema

- Genshin é um RPG com vários personagens.
- Assim como qualquer RPG, os personagens possuem uma ficha de atributos e vários fatores influenciam essa ficha.
- Dado um personagem com uma arma e talento específicos, então existe uma build que maximiza o número de output desse talento.
 - escolha de 5 artefatos do inventário
- O problema é otimizar uma build encontrando uma combinação ótima de artefatos.

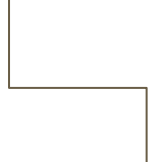
Espaço de Busca

No pior caso, um inventário cheio pode carregar 1500 artefatos.

Com ideal de 300 artefatos por slot, existem 300^5 combinações possíveis



tipo de artefato



$2,43e+12 = 2.430.000.000.000$
mais de 2 trilhões de possibilidades

Base de Dados

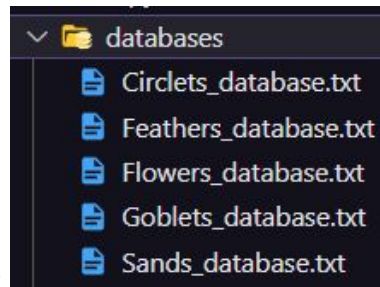
databases >  Circlets_database.txt

```
1 [31.10 CR]~[5.25 HP%]~[28.57 ATK%]~[6.22 CD]~[16.20 DEF]
2 [46.60 HP%]~[6.22 CD]~[13.99 ATK%]~[11.01 ER]~[31.12 ATK]
3 [187.00 EM]~[11.66 CD]~[9.72 CR]~[33.07 ATK]~[5.83 ER]
```

Representação de um artefato toString() da classe Artifact

Como não há uma base de dados pronta, então foi necessário gerar uma.

300 artefatos
em cada .txt



Como os dados foram gerados ?

Os artefatos foram gerados simulando o sistema de obtenção de artefatos no jogo.

Existem 5 métodos na classe Artifact para gerar cada tipo de artefato.

Limitação: Não foi considerado o sistema conjuntos de artefatos.

```
class Artifact:

    type_: str # 'Flower' | 'Feather' |
              # 'Sand' | 'Goblet' | 'Circlet'
    main_stat: MainStat
    sub_stats: list[SubStat, SubStat, SubStat, SubStat]
    mutation_rate: float # Usado apenas no ES
```

```
class MainStat:

    value: float
    type_: str # 'HP%' | 'HP' | 'ATK' |
              # 'ATK%' | 'DEF%' | 'EM' |
              # 'ER' | 'CR' | 'CD' |
              # 'Physical' | 'Anemo' | 'Geo' |
              # 'Electro' | 'hydro' | 'Pyro' |
              # 'Cryo' | 'Healing'
```

```
class SubStat:

    value: float
    type_: str # 'HP%' | 'HP' | 'ATK' |
              # 'ATK%' | 'DEF' | 'DEF%' |
              # 'EM' | 'ER' | 'CR' |
              # 'CD'
```

Como os dados foram gerados ?

Explicação detalhada da geração:

1. Escolhe um tipo de atributo principal + um valor de tabela para artefato 5★ nível máximo.
(esse atributo não pode se repetir nos subatributos)
2. Escolhe 4 tipos de subatributos e um valor inicial dado por uma tabela
(existem 4 possibilidades de valores iniciais para todos os subatributos)
3. Escolhe entre 3 e 4 *procs* de subatributo
4. Distribui os *procs* em cada 1 dos 4 subatributos
5. Soma os *procs* em cada subatributo considerando uma escolha aleatória das 4 possibilidades de valores de subatributo

Tabela com tipo e valor principal de artefatos do tipo tiara

Circlet of Logos	
Rarity	★★★★★
HP (%)	7.0 - 46.6
ATK (%)	7.0 - 46.6
DEF (%)	8.7 - 58.3
Elemental Mastery	28 - 186.5
CRIT Rate (%)	4.7 - 31.1
CRIT DMG (%)	9.3 - 62.2
Healing Bonus (%)	5.4 - 35.9

Possible sub-stat values ^[1]	
Rarity	★★★★★
HP	209.13 / 239.00 / 268.88 / 298.75
ATK	13.62 / 15.56 / 17.51 / 19.45
DEF	16.20 / 18.52 / 20.83 / 23.15
HP (%)	4.08 / 4.66 / 5.25 / 5.83
ATK (%)	4.08 / 4.66 / 5.25 / 5.83
DEF (%)	5.10 / 5.83 / 6.56 / 7.29
Elemental Mastery	16.32 / 18.65 / 20.98 / 23.31
Energy Recharge (%)	4.53 / 5.18 / 5.83 / 6.48
CRIT Rate (%)	2.72 / 3.11 / 3.50 / 3.89
CRIT DMG (%)	5.44 / 6.22 / 6.99 / 7.77

Como os dados foram gerados ?

```
"""
ban_substat -> substat that is already assigned in the main stat can't
be assigned again
returns 4 substats (Yield each substat in a generator fashion)
"""

@staticmethod
def generate_substats(ban_substat: str) -> Generator[SubStat]:
    filtered_substats = [
        substat for substat in POSSIBLE_SUB_STATS if substat != ban_substat
    ]
    rolls = 4 + random.randint(1, size=1) # 4 or 5 rolls
    substats_types: list[str] = list(
        random.choice(filtered_substats, 4, replace = False)
    )
    proc_types: list[str] = list(
        random.choice(substats_types, rolls, replace = True)
    )

    for substat_type in substats_types:
        procs = substats_types.count(substat_type) \
            + proc_types.count(substat_type)
        value = sum(random.choice(POSSIBLE_PROCS[substat_type], procs))
        yield SubStat(value, substat_type)
```

```
"""
n -> how many flowers are going to be generated
returns an instance of Artifact of type_ "Flower" with randomly generated
stats using Genshin Impact rules
"""

@staticmethod
def generate_flowers(n: int = 1) -> Artifact | list[Artifact]:
    flowers = []
    flower: Artifact

    for _ in range(n):
        ban = 'HP'
        substats = [substat for substat in Artifact.generate_substats(ban)]

        flower = Artifact('Flower', MainStat(4780, 'HP'), substats)
        flowers.append(flower)

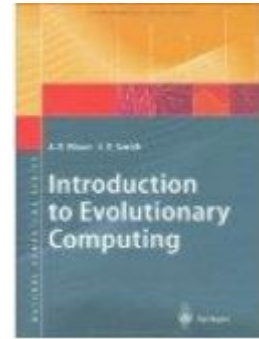
    return flowers if n > 1 else flower
```

1 dos 5 métodos para gerar artefatos

Modelagem do

1. Representação
2. Recombinação
3. Mutação (no GA)
4. Mutação (no ES)
5. Fitness
6. Seleção dos pais e dos sobreviventes
7. Parâmetros

problema



Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional - implemented by Roulette Wheel
Survival selection	Generational

Table 6.1. Sketch of the simple GA

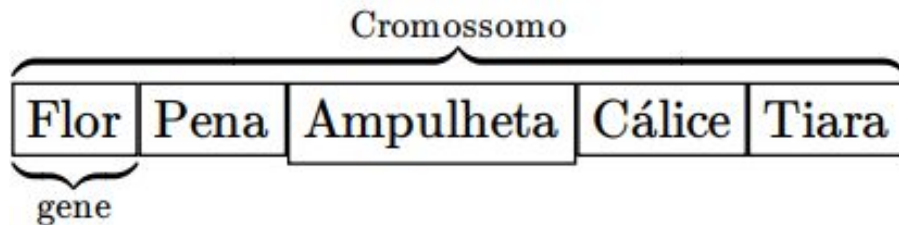
Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	Deterministic elitist replacement by (μ, λ) or $(\mu + \lambda)$
Speciality	Self-adaptation of mutation step sizes

Table 6.2. Sketch of ES

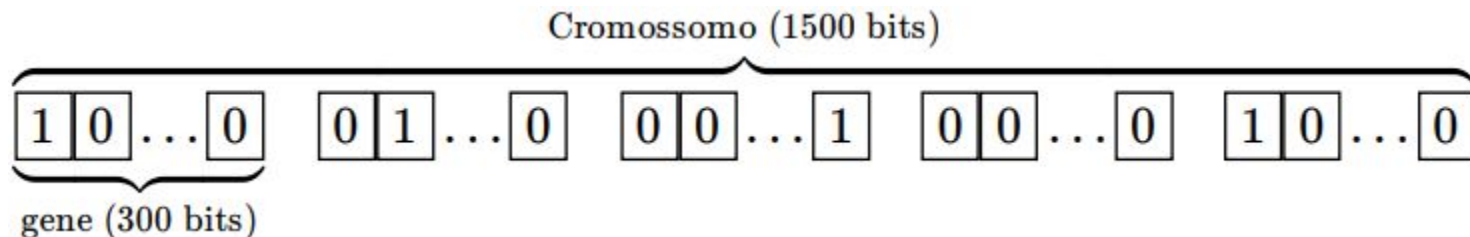
Representação

Em ambos os algoritmos, um cromossomo é uma lista de artefatos (build) e os genes são os artefatos.

É possível mapear uma lista de artefatos para a representação **array de bits**.



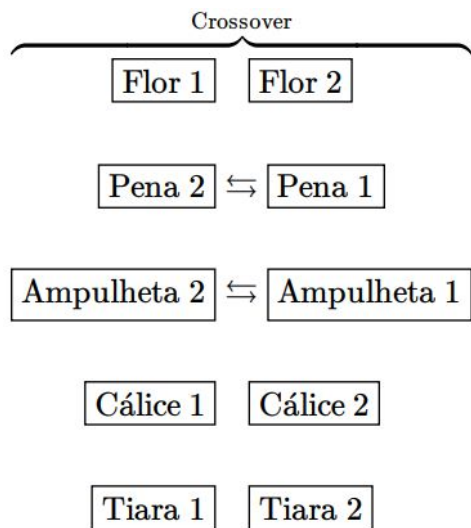
Não é possível mapear uma lista de artefatos para a representação **vetor de reais**, pois um gene não possui as mesmas propriedades que um número real. Será necessário adaptar os componentes do algoritmo ES.



Recombinação

Ambos os algoritmos usaram o N-point crossover.

N = 1 até 5.

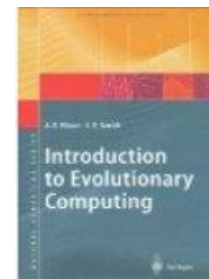


GA	ES
3-Point	2-Point

Representation	Real-valued vectors
Recombination	Discrete or intermediary

O livro explica que o ES utiliza **recombinação discreta** ou **intermediária**. Para a representação da lista de artefatos, a recombinação discreta é igual a 5-point crossover, mas com o detalhe que somente 1 dos 2 cromossomos é escolhido como filho.

Com o objetivo de gerar 2 filhos e testar outras possibilidades de recombinação, foi utilizado o 2-point crossover na ES.



Recombinação

```
102 # N-point crossover -> Escolhe N pontos de corte e gera 2 novos indivíduos
103 def crossover(b1: Build, b2: Build, points: int = 2) -> tuple[Build, Build]:
104
105     # Link que ajuda a entender o código
106     # https://medium.com/@samiran.bera/crossover-operator-the-heart-of-genetic-algorithm-6c0fdcb405c0
107
108     possible_cuts = [0, 1, 2, 3, 4]
109     random_cut = np.random.choice(possible_cuts, size = points, replace = False)
110
111     #           <--0           <--1           <--2           <--3           <--4
112     cromossome_1 = [b1["flower"], b1["feather"], b1["sand"], b1["goblet"], b1["circlet"]]
113     cromossome_2 = [b2["flower"], b2["feather"], b2["sand"], b2["goblet"], b2["circlet"]]
114
115     for cut in np.sort(random_cut):
116         temp = cromossome_1[:cut] + cromossome_2[cut:]
117         cromossome_2 = cromossome_2[:cut] + cromossome_1[cut:]
118         cromossome_1 = temp
119
120     b1 = Build(cromossome_1[0], cromossome_1[1], cromossome_1[2], cromossome_1[3], cromossome_1[4])
121     b2 = Build(cromossome_2[0], cromossome_2[1], cromossome_2[2], cromossome_2[3], cromossome_2[4])
122
123     return (b1, b2)
```

Mutação (no GA)

Mutação

Flor

Pena

Ampulheta

Cálice

Tiara da base de artefatos \leftrightarrow Tiara

Mutation

Bit flip

1 0 ... 0

gene antes

0 1 ... 0

gene depois

Uma mutação bit-flip em uma lista de artefatos é feita trocando um artefato por outro artefato da base de dados.

Na versão final, foi adicionado uma heurística na mutação, bit flip obrigatório em cada slot de artefato e apenas os **não selecionados** passam por crossover.

Mutação (no GA)

```
127 # Reset mutation -> Substitui um gene por outro gerado aleatoriamente
128 def mutation(b: Build, bag_of: dict[str, np.ndarray], useless_substats: list[str]) -> Build:
129
130     genes_types = ["flower", "feather", "sand", "goblet", "circlet"]
131
132     for gene_type in genes_types:
133
134         new_artifact = b[gene_type]
135
136         useless_stat_count = 3
137         while useless_stat_count > 2:
138             new_artifact = np.random.choice(bag_of[gene_type], size = 1, replace = False)[0]
139             useless_stat_count = new_artifact.count_useless_substats(useless_substats)
140
141         # b[random_type] = new_artifact
142         b[gene_type] = new_artifact
143
144     return b
```

Mutação (no ES) ← adaptada para lista de artefato

Nos algoritmos ES atuais são feitas duas perturbações gaussianas (mutações). A primeira é no desvio padrão do gene (cada gene carrega seu próprio desvio padrão) e a segunda é no gene (essa mutação usa o desvio padrão na fórmula de Curva de Gauss).

Como a perturbação gaussiana só é definida para a representação vetor de reais, então foi preciso adaptar esse componente.

Cada gene (artefato) carrega um desvio padrão inicial de 2.5 e não é feita nenhuma mutação nesse valor.

Mutação (no ES) ← adaptada para lista de artefato

Primeiro foi decidido que é interessante trocar os tipos dos stats do artefato.

Em segundo lugar foi definido a diferença Δ_{artefato} entre dois artefatos.

Por fim, foi usada a fórmula da curva de Gauss escrita no livro (com algumas alterações) para encontrar a probabilidade de aceitar a perturbação Δ_{artefato} .

```
"""
self -> current artifact to be used to calculate the difference
other -> other artifact to calculate difference between them
returns the amount of different types_
"""
def difference(self, other: Artifact) -> int:

    ammount_of_diferences = 0

    if self.main_stat.type_ == other.main_stat.type_:
        ammount_of_diferences += 1

    for self_substat, other_substat in zip(self.sub_stats, other.sub_stats):
        if self_substat.type_ == other_substat.type_:
            ammount_of_diferences += 1

    return ammount_of_diferences # retorna 0 até 5
```

$$\Delta_{\text{artefato}} = \text{artefato}_1 - \text{artefato}_2 = \text{quantidade de tipos de stats diferentes}$$

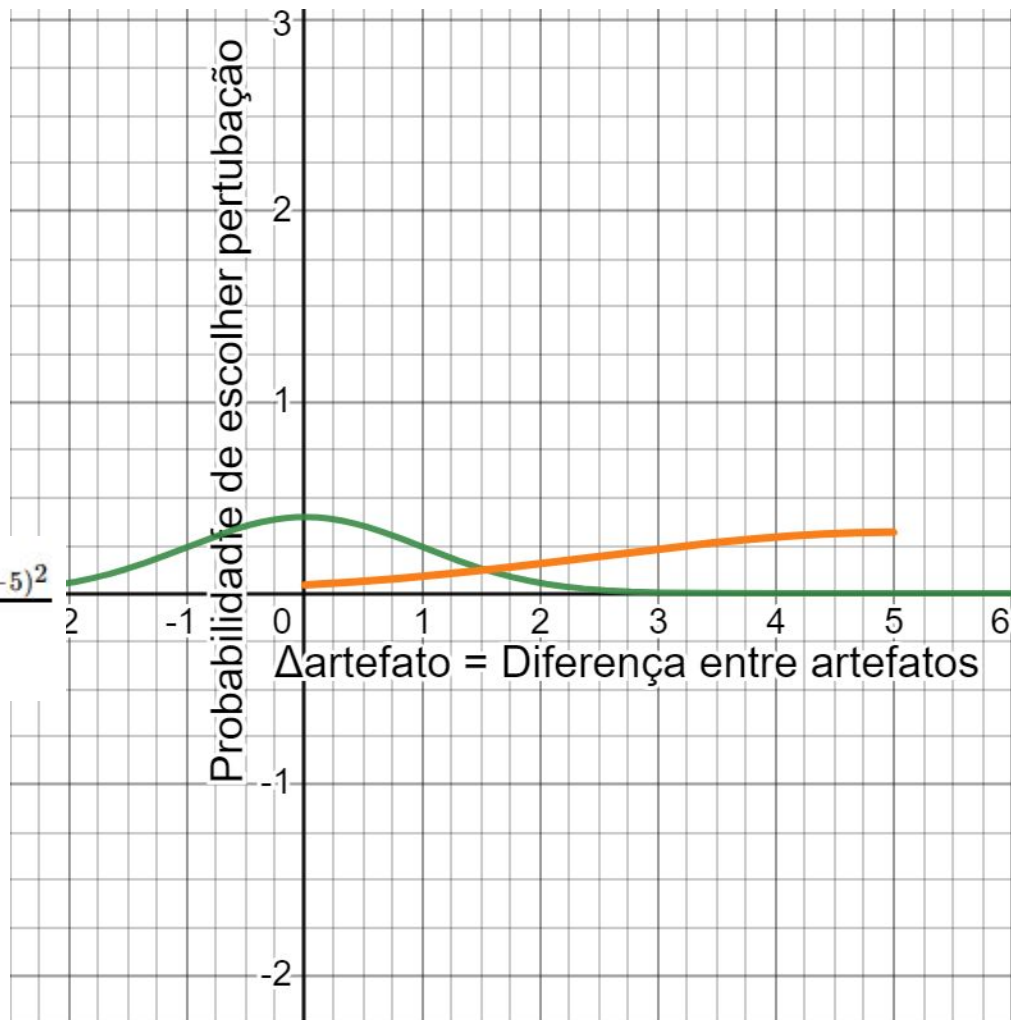
Mutação (no ES)

$$p(\Delta x_i) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(\Delta x_i - \xi)^2}{2\sigma^2}}$$

Distribuição de Gauss do livro.
 σ = desvio padrão = 1 e ξ = média = 0

$$p(\Delta \text{artefato}) = 2 \cdot \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(\Delta \text{artefato} - 5)^2}{2\sigma^2}}$$

Distribuição de Gauss usada.
 σ = **mutation rate** = 2.5
 ξ = **média** = 5



Visualização interativa:

[Algoritmo Evolucionário - ES \(desmos.com\)](https://www.desmos.com/algebra)

Mutação (no ES)

```
# Gaussian Perturbation -> Substitui cada gene por outro gerado aleatoriamente usando a chance de Gauss
def mutation(b: Build, bag_of: dict[str, np.ndarray]) -> Build:

    genes_types = ["flower", "feather", "sand", "goblet", "circlet"]
    for gene_type in genes_types: # Faz um perturbação em cada gene
        while True: # Para quando ocorre uma perturbação bem sucedida
            artifacts = bag_of[gene_type]
            new_artifact = np.random.choice(artifacts, size = 1, replace = False)[0]

            delta_artifact = b[gene_type].difference(new_artifact)

            probability = 2 * \
                np.divide(
                    1.0, b[gene_type].mutation_rate * np.sqrt(2*np.pi)
                ) * \
                np.power(
                    np.e, -np.divide(np.power(delta_artifact - 5, 2),
                    (2 * np.power(b[gene_type].mutation_rate, 2)))
                )

            if np.random.rand(1) <= probability:
                b[gene_type] = new_artifact
                break

    return b
```

Fitness

Site com as fórmulas do jogo:

[Artifact Optimizer for Genshin Impact](#)

[Damage Formula - Theorcrafting Library - KQM \(keqingmains.com\)](#)

Para cada combinação de personagem, arma e talento, existe uma função única de fitness.

Nesse projeto, foram consideradas 2 funções fitness.

Ambas as funções fitness utilizam a ficha do personagem para calcular um valor final.

Fitness 1

Hu Tao + Báculo de Homa R1 + Habilidade Elemental
+ Ataque Carregado

$$\text{fitness}(\text{Artifacts}) = 242.6\% * \text{Total_ATK} * (100\% + \text{Total_Crit_Rate} * \text{Total_Crit_DMG}) \\ * (100\% + \text{Total_DMG_Bonus}) * \text{Enemy DEF Multiplier} * (100\% - \text{Enemy_Pyro_Res.})$$

$$\text{Total_ATK} = \text{Elemental_Skill_ATK} + \text{Total_Weapon_Passive_ATK} + \text{Base_ATK} \\ * (100\% + \text{Artifacts_ATK}\%) + \text{Artifacts_ATK_flat}$$

$$\text{Total_Crit_Rate} = 5\% + \text{Artifacts}$$

$$\text{Total_Crit_DMG} = 50\% + \text{Char_Crit_DMG} + \text{Weapon_Crit_DMG} + \text{Artifacts}$$

$$\text{Total_DMG_Bonus} = \text{Total_Pyro_DMG_Bonus} + \text{Artifact_Active_Set_Bonus}$$

$$\text{Artifact_Active_Set_Bonus} = 22.5 = 15\% + 7.5\%$$

$$\text{Total_Pyro_DMG_Bonus} = \text{Active_Talent_Passive} + \text{Artifacts_Pyro}$$

$$\text{Active_Talent_Passive} = 33\%$$

$$\text{Enemy_DEF_Multiplier} = 0.5 = (\text{Char_Level} + 100) / (\text{Char_Level} + 100 + \text{Enemy_Level} + 100)$$

$$\text{Char_Level} = 90$$

$$\text{Enemy_Level} = 90$$

$$\text{Enemy_Pyro_Res.} = 10\%$$

$$\text{Elemental_Skill_ATK} = \text{Total_HP}$$

$$\text{Total_Weapon_Passive_ATK} = \text{Weapon_Passive_1_ATK} + \text{Weapon_Passive_2_ATK}$$

$$\text{Weapon_Passive_1_ATK} = 0.8\% * \text{Total_HP}$$

$$\text{Weapon_Passive_2_ATK} = 1\% * \text{Total_HP}$$

$$\text{Total_HP} = \text{Char_HP} * (100\% + \text{Weapon_Passive_1_HP} + \text{Artifacts_HP}\%) + \text{Artifacts_HP_flat}$$

$$\text{Char_HP} = 15552$$

$$\text{Weapon_Passive_1_HP} = 20\%$$

$$\text{Base_ATK} = \text{Weapon_Base_ATK} + \text{Char. Base_ATK}$$

$$\text{Weapon_Base_ATK} = 608$$

$$\text{Char_Base_ATK} = 106.51$$

$$\text{Char_Crit_DMG} = 38.4\%$$

$$\text{Weapon_Crit_DMG} = 66.2\%$$

Fitness 2

Zhongli + Borla Preta R5 + Dano de Absorção do

$$\text{fitness}(\text{Artifacts}) = 150\% * (23\% * \text{Total_HP} + 2711.5) * (100\% \\ + \text{Total_Shield_Strength})$$

$$\text{Total HP} = \text{char.HP} * (100\% + \text{Weapon_Passive} + \text{Artifacts_HP}\%) \\ + \text{Artifacts_Hp_flat}$$

$$\text{Total_Shield_Strength} = 25\% + \text{Artifacts}$$

$$\text{Weapon_Passive} = 46.9\%$$

$$\text{char. HP} = 14695$$



Build.py

[github Build.py](#)

```
class Build(dict):
```

```
class Sheet(dict):
```

```
def get_artifact_sheet(self) -> Sheet:
```

Seleção dos pais e dos sobreviventes

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional - implemented by Roulette Wheel
Survival selection	Generational

Table 6.1. Sketch of the simple GA

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	Deterministic elitist replacement by (μ, λ) or $(\mu + \lambda)$
Speciality	Self-adaptation of mutation step sizes

Table 6.2. Sketch of ES

	GA	ES $\mu + \lambda$
Seleção dos pais (para aplicar apenas crossover)	Roleta proporcional ao fitness	Seleção uniforme aleatória
Seleção dos sobreviventes	Nova geração sobrescreve a geração atual	$\mu = \lambda$, Metade mais apta sobrevive

Roleta do GA

```
# Seleção por roleta que seleciona 50% (half_cut) dos cromossomos
selected_mask = np.full(POP_SIZE, False, dtype = bool)
unique_indexes = set()
while len(unique_indexes) < half_cut:
    random_num = np.random.rand(1)[0]
    temp = df["Normalized_Fitness_Cummulative_Sum"]
    temp = temp.append(pd.Series(random_num))
    temp.sort_values(ascending = True, inplace = True)
    temp.reset_index(drop = True, inplace = True)
    selected_index = np.where(temp.values == random_num)[0][0]

    selected_mask[selected_index] = True
    unique_indexes.add(selected_index)

# Aplica crossover nos selecionados
selected_indexes = np.where(selected_mask == True)[0]

# Aplica mutação nos não selecionados pela roleta
not_selected_indexes = np.where(selected_mask == False)[0]
```

Seleção uniforme aleatória

```
# Seleciona aleatoriamente e uniformemente os  $\mu$  parents  
selected = np.random.choice(df["Cromossomes"], size = MU, replace = False)
```


Parâmetros

```
@Timer(name="decorator",
      text="Tempo da busca: {:.4f} segundos")
def GA(fitness: Callable, |
      target_fitness: int,
      useless: list[str],
      seed: int):
```

Parâmetros

RANDOM_SEED = seed

POP_SIZE = 300

MAX_GENERATIONS = 10000

MAX_NO_CHANGE_GENERATIONS = 750

CROSSOVER_RATE = 0.75

MUTATION_RATE = 0.50

N_POINT = 3 # N-point crossover

```
@Timer(name="decorator",
      text="Tempo da busca: {:.4f} segundos")
def ES_multimember_plus(fitness: Callable,
                       target_fitness: int,
                       seed: int):
```

Parâmetros

RANDOM_SEED = seed

POP_SIZE = 300

MAX_GENERATIONS = 10000

MAX_NO_CHANGE_GENERATIONS = 750

CROSSOVER_RATE = 0.50

MUTATION_RATE = 2.5 # Desvio padrão

N_POINT = 2 # 2-point crossover

MU: int = POP_SIZE # μ -> número de indivíduos selecionados

LAMBDA: int = POP_SIZE # λ -> Número de filhos gerados

Experimentos

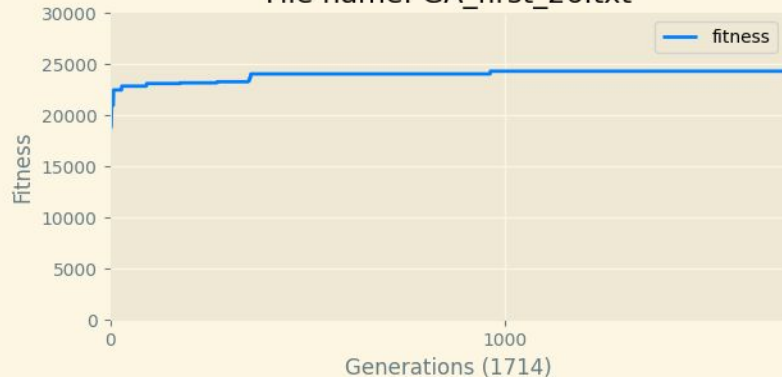
Experimentos

GA		ES	
fitness 1	fitness 2	fitness 1	fitness 2
30	30	30	30
Total: 120 experimentos			

Melhor amostra de cada algoritmo com fitness 1 e fitness 2

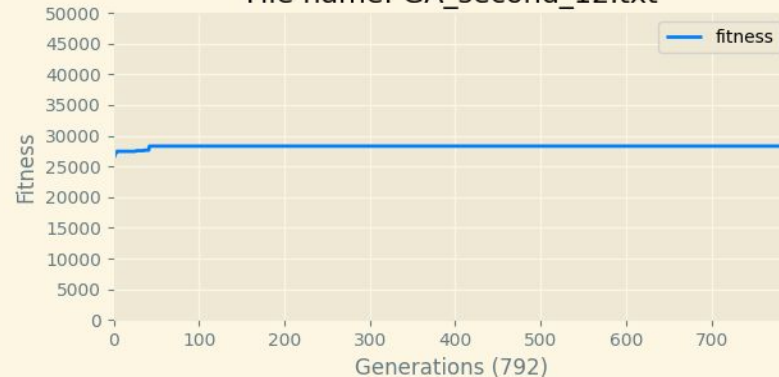
Best Fitness Over Generations

File name: GA_first_26.txt



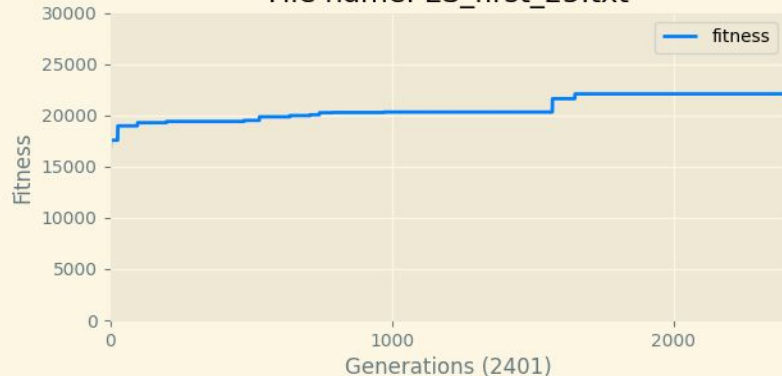
Best Fitness Over Generations

File name: GA_second_12.txt



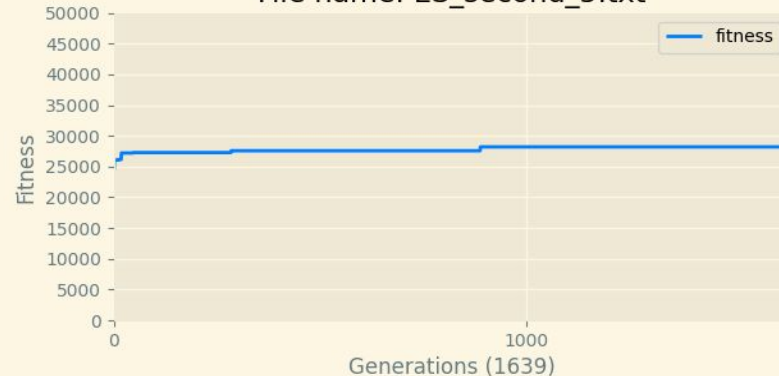
Best Fitness Over Generations

File name: ES_first_25.txt



Best Fitness Over Generations

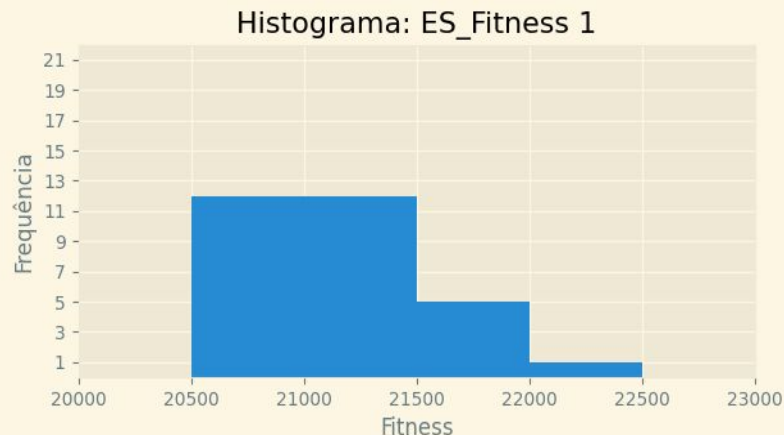
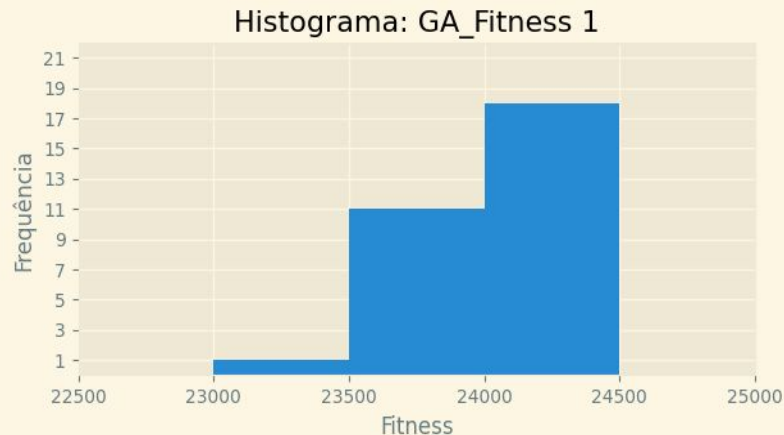
File name: ES_second_5.txt



Comparação

A pior execução de **GA** com **fitness 1**, foi encontrado um indivíduo com, no mínimo, 23000 de fitness.

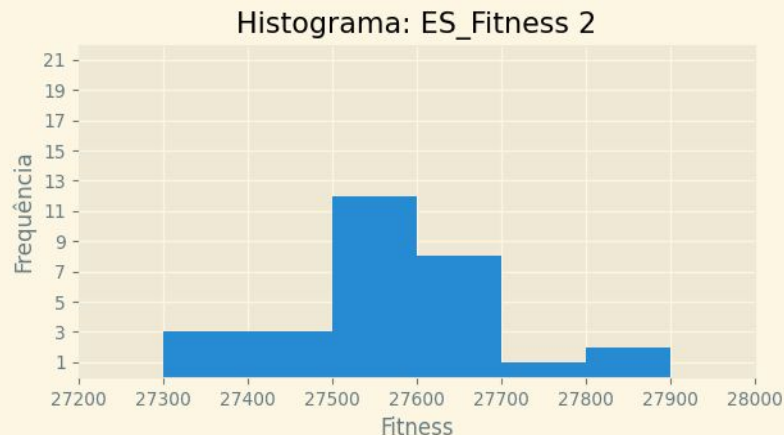
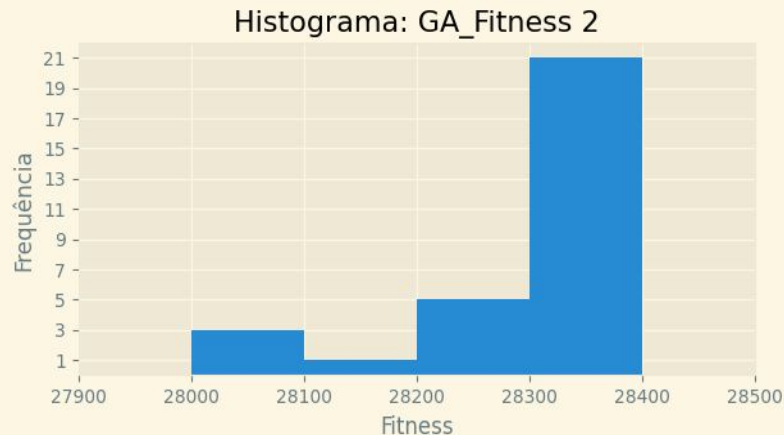
Como todas as amostras da ES encontraram fitness menor que 23000, então o **GA obteve melhores resultados**.



Comparação

A pior execução de **GA** com **fitness 2**, foi encontrado um indivíduo com, no mínimo, 28000 de fitness.

Como todas as amostras da ES encontraram fitness menor que 27900, então o **GA obteve melhores resultados**.



Principais 3 Referências

Purba, K.R. (2015). **Optimization of Auto Equip Function in Role-Playing Game Based on Standard Deviation of Character's Stats Using Genetic Algorithm**. In: Intan, R., Chi, CH., Palit, H., Santoso, L. (eds) Intelligence in the Era of Big Data. ICSIIIT 2015. Communications in Computer and Information Science, vol 516. Springer, Berlin, Heidelberg.
https://doi.org/10.1007/978-3-662-46742-8_6

→ Tema do projeto (Otimização Combinatória) inspirado nesse artigo.

Rodrigo F. C., Gilson P. dos S. J., Lauro B. F., Marília dos A. S., Brunna L. C. da S. (2019). **Geração automática de horário escolar com algoritmo genético**. Edição v. 8 n. 2: Revista Eixo. Seção: Artigos

→ Mostra como é possível considerar restrições hard e soft na função fitness e que a taxa de mutação com valores altos é interessante para explorar esse tipo de espaço de busca.
Fitness alternativo baseado nesse artigo: permitir apenas builds com um mínimo e máximo de algum atributo usando restrições hard e soft.

Montini, Beatriz de Barros. **Algoritmo genético para Problema Generalizado de Atribuição**. Universidade Estadual Paulista (Unesp), 2022. Disponível em: <<http://hdl.handle.net/11449/217533>>.

→ Mostra como representar um cromossomo do problema da mochila em uma string de bits.
Esse artigo inspirou o mapeamento de array de bits para lista de artefatos

Referências extras

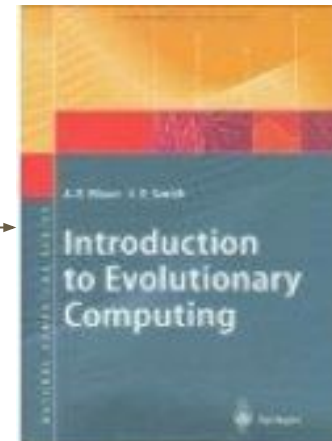
[MIPT: desenvolvimento de um webservice que gera monstros ideais para mestres de role play game \(RPG\). \(pucgoias.edu.br\)](http://pucgoias.edu.br)

[Geração Adaptativa de Personagens para Role Playing Games \(usp.br\)](http://usp.br)

Evolution Strategies Nikolaus Hansen, Dirk V. Arnold and Anne Auger February 11, 2015
<http://www.cmap.polytechnique.fr/~nikolaus.hansen/es-overview-2015.pdf>

Genetic Algorithm in Python generates Music
<https://www.youtube.com/watch?v=aOsET8KapQQ>

[EIBEN, A. E.; SMITH, J. E. Introduction to Evolutionary Computing. Berlin Springer, 2003](#)





FIM

