**Bruno Fernandes Bessa de Oliveira**

**Professor Alexandre Cláudio Botazzo Belbem**

**MAI5030 Introdução aos Algoritmos Evolutivos**

**8 de Maio de 2020**

**Resumo**

Este relatório organiza as atividades práticas decorridas no primeiro bimestre da disciplina de Introdução aos Algoritmos Evolutivos, ministrada no primeiro semestre de 2020 no Instituto de Ciências Matemáticas e Computação da USP no campus de São Carlos.

## 1 Introdução

Os algoritmos evolutivos parte do que chamamos Computação Bio-Inspirada. Neste domínio da Ciência da Computação busca-se criar algoritmos que reproduzam mecanismos observados na natureza. Mais especificamente, algoritmos evolutivos aplicam os conceitos encontrados na Teoria da Evolução das Espécies por meio de seleção natural, desenvolvida primeiramente pelo notório naturalista britânico Charles Darwin em meados do século XIX. Com estes algoritmos simula-se processos evolutivos em populações de indivíduos.

Nestes programas os indivíduos da população são representados por uma máquina de estados finita que processa uma sequência de símbolos. A representação dos indivíduos se dá por um conjunto de variáveis numéricas que assumem os valores possíveis de um gene. O conjunto de **genes** de um indivíduo é denominado **cromossomo**. A partir de uma população inicial são usados operadores de reprodução para gerar novas populações descendentes bem como uma função de aptidão (**fitness**),

que representa a pressão seletiva do ambiente. Estes elementos de um algoritmo evolutivo são descritos a seguir.

- **Mutação:** é um operador de reprodução que modifica aleatoriamente um ou mais genes de um cromossomo. Com este operador um indivíduo gera uma cópia de si mesmo (reprodução assexuada), que pode conter alterações. A taxa de incidência de mutações é geralmente pequena.
- **Recombinação**: é um operador de reprodução que representa a reprodução sexuada. Por meio deste operador novos indivíduos são gerados a partir da troca de fragmentos dos cromossomos de um ou mais indivíduos de todo que o novo indivíduo possui características de ambos os seus genitores, aumentando a variabilidade genética da nova população.
- **Função de aptidão (fitness):** é a medida numérica da adequação de um indivíduo (por extensão, de uma população). A seleção de indivíduos é feita com base em sua aptidão.

## 2 - Metodologia

As atividades práticas propostas foram desenvolvidas na linguagem de programação Python 3 com uso de uma biblioteca de código aberto chamada pyeasyga, de autoria do desenvolvedor Ayodeji Remi-Omosowon. Esta biblioteca permite que modifiquemos a estrutura básica da estratégia evolutiva para, por exemplo, indicar qual a função de aptidão que desejamos implementar.

Também realizamos a implementação completa do algoritmo evolutivo para o problema “one max” utilizando somente a biblioteca padrão da linguagem Python. Este código está disponível no documento relacionado abaixo.

`one_max_problem/python_puro_sga_one_max.py`

Para cada solução proposta, os resultados exibidos são:

- média de 10 execuções para cada ponto na curva no caso de função fitness
- matriz de covariância da melhor população da última geração produzida nos casos de problemas com números reais

**3 - SGA e CGA**

Implementamos para uma coleção de problemas selecionados, dois algoritmos genéticos. São eles:

- **sGA (simple genetic algorithm):** é um algoritmo utilizado para fins didáticos. Ele segue a seguinte estrutura básica: uma população inicial é gerada **aleatoriamente**. Cada indivíduo é medido quanto à função de aptidão. Através de torneios, selecionamos indivíduos bem avaliados para que sejam os genitores de uma nova população. Sempre que indivíduos novos são criados, há chance de haver mutação. Este processo se repete até que um critério de parada seja atingido
- **cGA (compact genetic algorithm):** diferentemente do sGa que usa mutação e recombinação, neste algoritmo utiliza um vetor de probabilidades de que os indivíduos da geração atual produza descendentes. Esta implementação requer menos recursos computacionais.

**4 - Problema one max**

O problema one max consiste na otimização da função abaixo onde ***x*** é o vetor numérico que corresponde ao cromossomo. Cada posição do cromossomo pode assumir somente os valores 0 e 1, de modo que esta função conta quantos genes (bits) são iguais a 1.

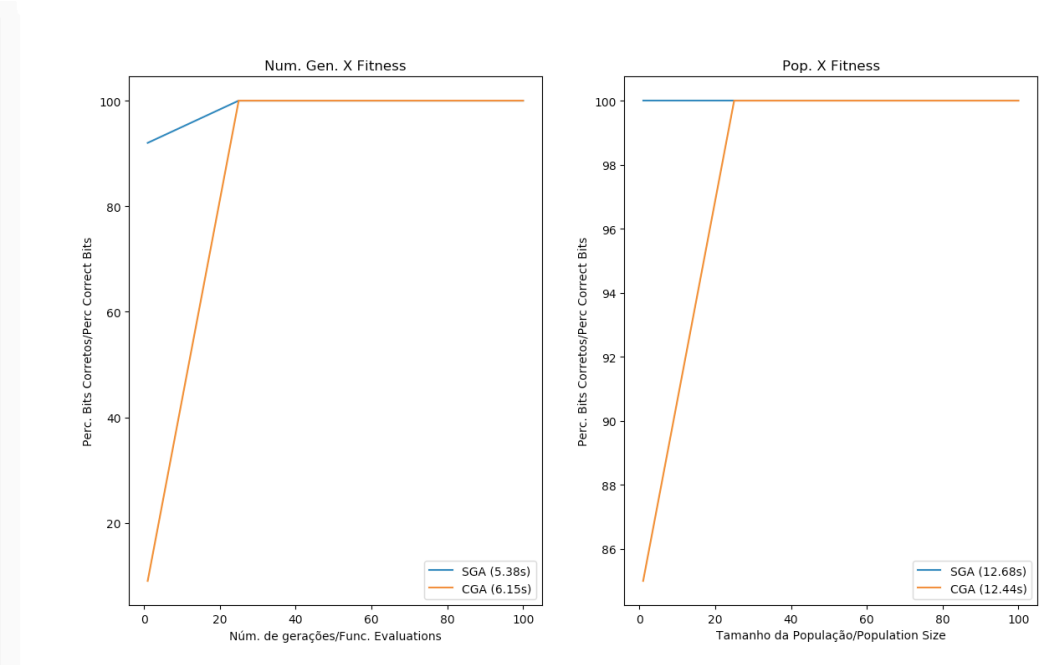
$$f(\overrightarrow{x}) = \sum_i x_i$$

A implementação desta solução se encontra nos arquivos relacionados abaixo.

`one_max_problem/sga_one_max.py`

`one_max_problem/cga_one_max.py`

Os resultados obtidos através da execução do programa são exibidos abaixo.




Podemos ver que o cGA converge muito rapidamente para a solução desejada, conforme aumentamos o número de execuções (gerações) e também o tamanho da população.

## 5 - Problemas Trap-5 e InvTrap-5

A função Ttrap-5 trata-se de um problema artificial que consiste em dividir o cromossomo binário candidato em partições disjuntas de 5 genes cada. Este particionamento permanece fixo durante todo o processo de otimização. A cada grupo de 5 bits é aplicada a função

$$f(u)\begin{cases}5, & \text{se } u = 5\\ 4-u, & \text{se } u < 5\end{cases}$$

que possui $2^{n/5}$ ótimos locais, nos quais a função fica aprisionada. Seu valor ótimo global é aquele em que todos os bits dos blocos são iguais a 1. De forma semelhante trabalhamos a função InvTrap-5, dada abaixo também foi explorada

$$f(u)\begin{cases}5, & \text{se } u = 0\\ u-1, & \text{se } u > 0\end{cases}$$

As implementações destes problemas se encontram nos arquivos

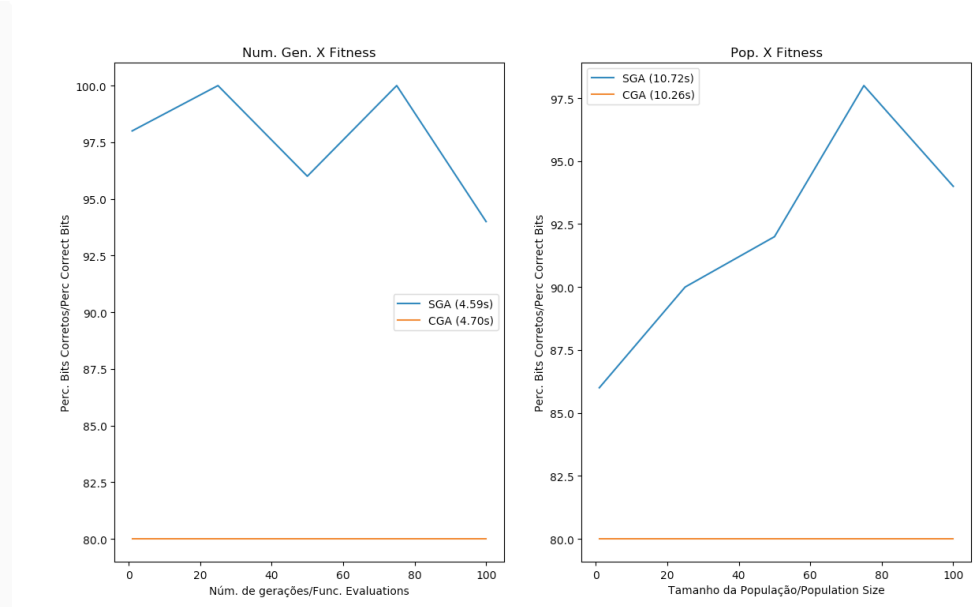
`trap5_problem/sga_trap.py`

`trap5_problem/sga_trap.py`

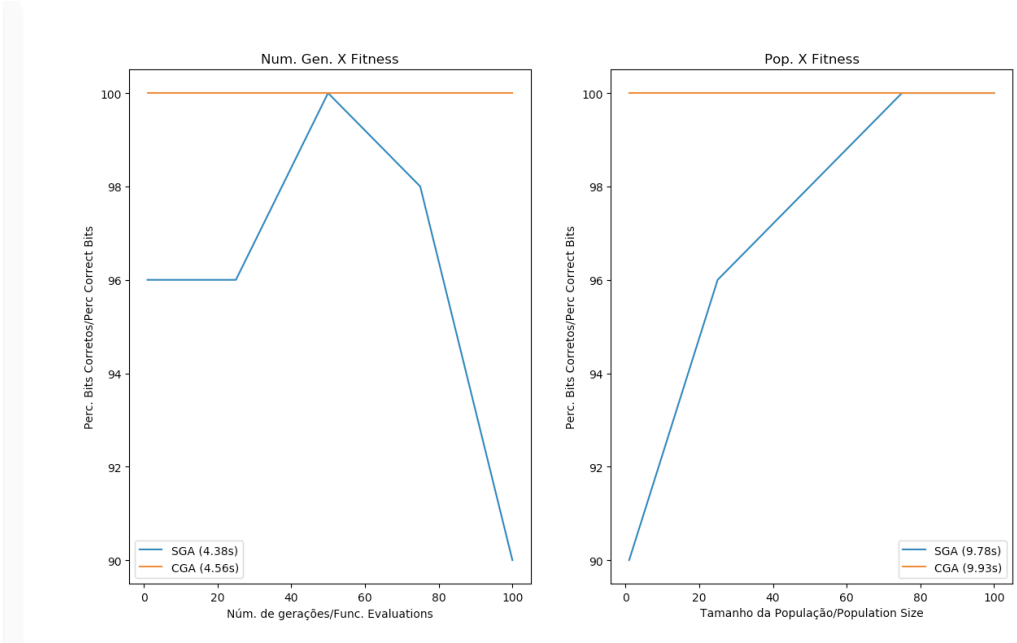
`invtrap5_problem/sga_invtrap.py`

`invtrap5_problem/sga_invtrap.py`

e os resultados obtidos são exibidos abaixo. Para Trap-5:



E para InvTrap-5:




Com estes resultados podemos ver a convergência estável do cGA frente ao sGA.

## 6 - Problema de números reais "sphere"

Para este problema, bem como os demais problemas com números reais, geramos valores escalares para cada posição do cromossomo (de acordo com sGA ou cGA). A função de aptidão deste problema é uma parábola no espaço n-dimensional, dada por:

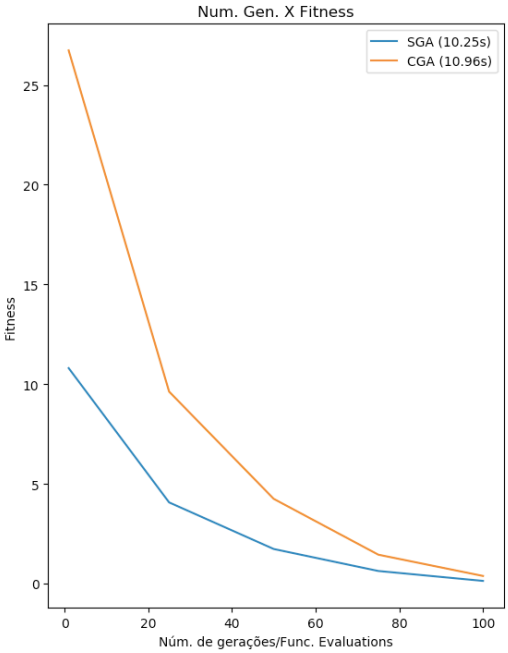
$$f(\vec{x}) = \sum_i x_i^2$$

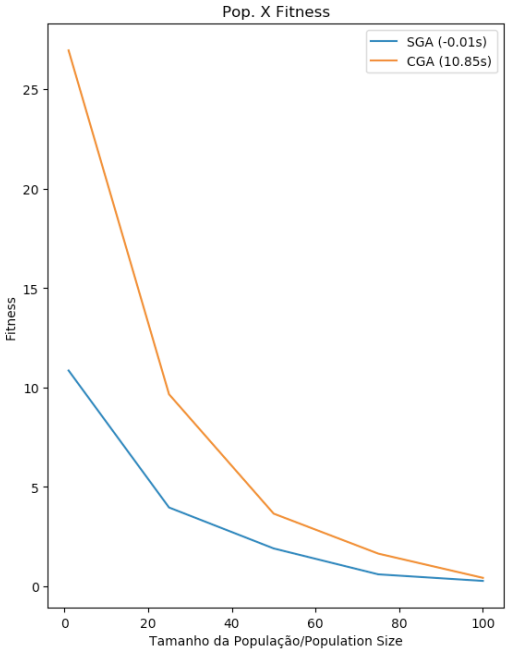
A implementação desta solução se encontra nos arquivos relacionados abaixo.

`real_number_problem/sga_real_number.py`

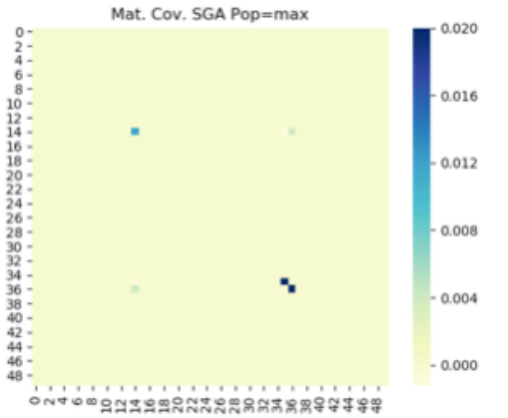
`real_number_problem/cga_real_number.py`

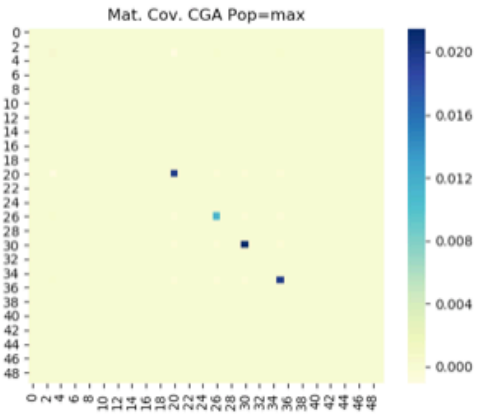
Os resultados obtidos através da execução do programa são exibidos abaixo.




As matrizes de covariância para uma população de 100 indivíduos:




Vemos que há uma baixa correlação conforme o número de gerações aumenta. Também observamos que a performance (fitness) da população reduz mais lentamente para o cGA do que para o sGA com o aumento do número de gerações.

### 7 - Problema de números reais "Rosenbrock"

A função Rosenbrock é um problema de optimização emblemático pois possui um ótimo global cercado por muitos ótimos locais. É dada pela expressão

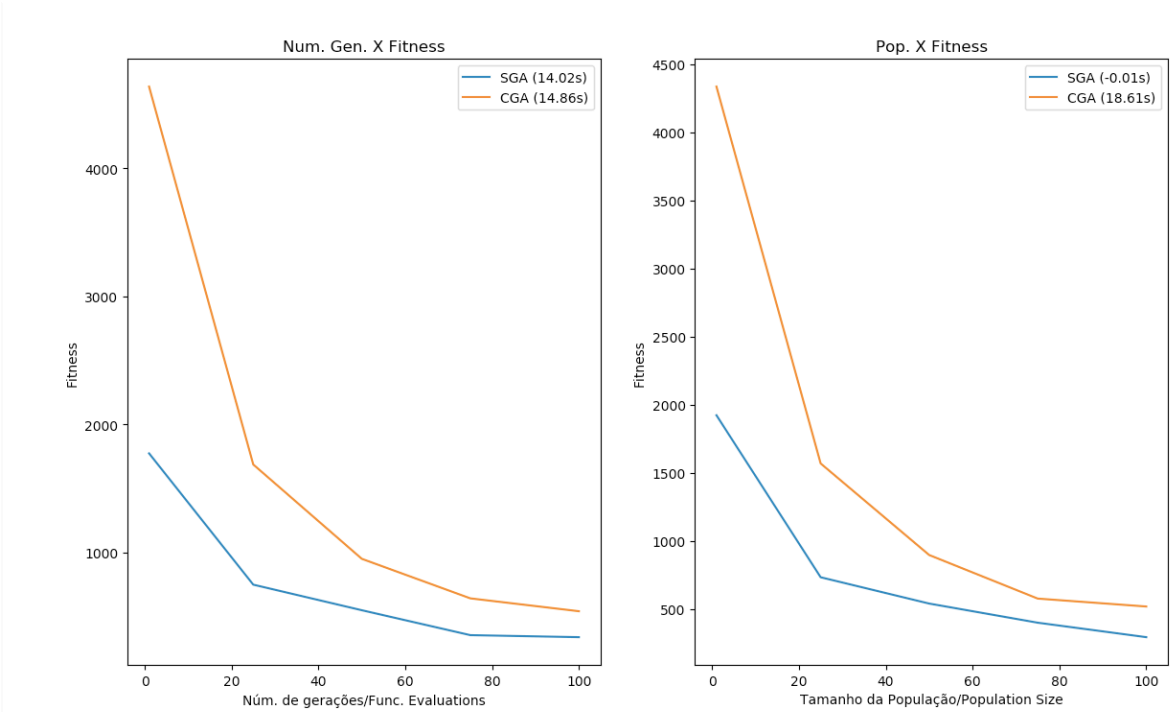
$$f(x) = \sum_{i=1}^{d-1} 100(x_{i-1} - x_i^2) + (x_i - 1)^2$$

A implementação desta solução se encontra nos arquivos relacionados abaixo.

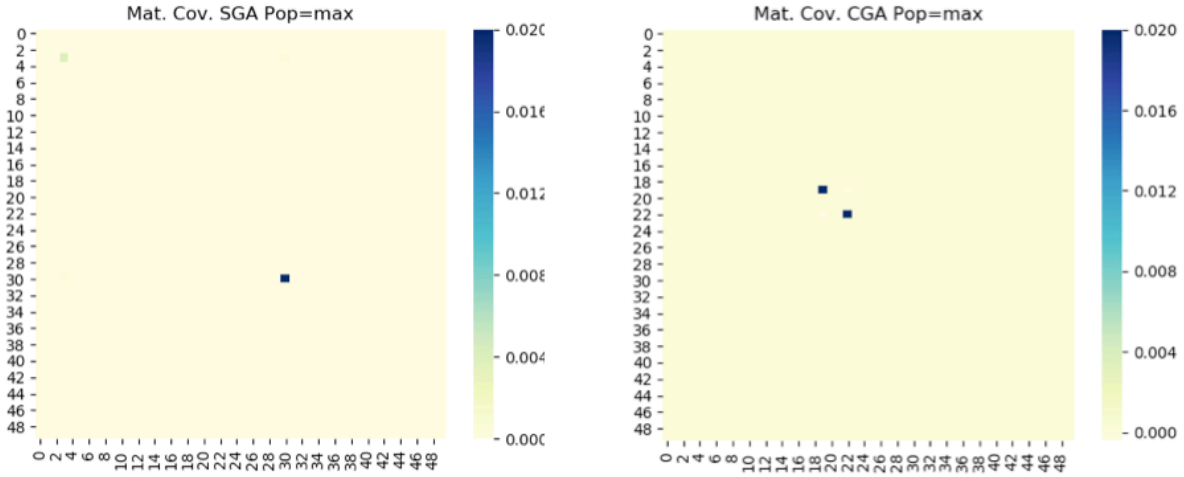
`rosenbrock_problem/sga_rosenbrock.py`

`rosenbrock_problem/cga_rosenbrock.py`

E o resultado da execução é dado abaixo.



Bem como o estudo de covariância para os genes da melhor população da última geração (100).




Em que novamente vemos a baixa covariância entre s genes.

## 8 - Problema de otimização multi-objetivo

Otimização multi-objetivo é a busca de otimização de mais de uma função de adequação ao mesmo tempo. Neste cenário, o conceito de solução ótima dá lugar ao conceito de solução **não dominada**. Uma solução não dominada (também chamada ótimo de Pareto) caracteriza-se por não haver uma outra solução admissível que melhore simultaneamente todos os objetivos (funções de adequação) sem que haja a degradação de ao menos um deles.

Para explorar esta form de otimização escolhemos  problema **NSGA-II** (Non-dominated Sorting Genetic Algorithm II), proposto por Deb et al em 2002. O NSGA-II possui um esquema de seleção da nova população baseado em dois principais operadores: a **Ordenação de Pareto** e uma métrica de promoção de diversidade da população que é livre de parâmetros, chamada **crowding distance** (CD).

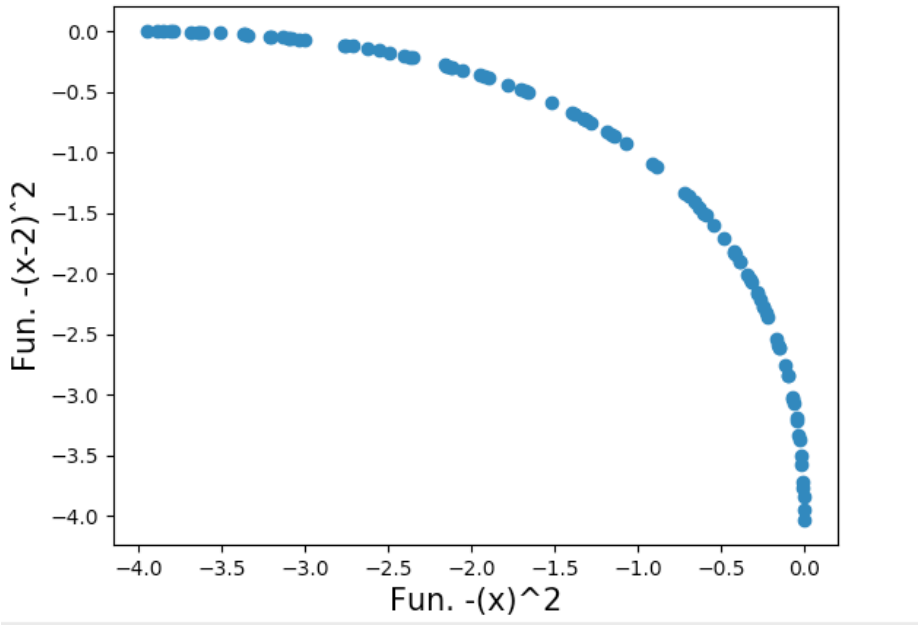
- **Ordenação de Pareto (non-dominated Sorting)**: consiste em classificar uma população P em d subconjuntos tais que $F_1$ = {indivíduos não-dominados de P e que não são dominados por nenhum outro de P } (rank 1), $F_2$ = {indivíduos não-dominados de P\F1 e que não são dominados por nenhum outro de P\$F_2$} (rank 2). Assim sucessivamente até termos $F_d$ = {indivíduos não-dominados de P\{ $F_1 \bigcup F_2 \ldots \bigcup F_{d-1}$} e que não são dominados por nenhum outro de P\{ $F_1 \bigcup F_2 \ldots \bigcup F_{d-1}$}, rank d, onde P = $F_1 \bigcup F_2 \ldots \bigcup F_d$.

- **Crowding Distance**: é a distância de uma solução com relação às demais em um mesmo rank no espaço das funções objetivo. Quanto maior o valor de CD, mais distante a solução se encontra de suas vizinhas, tornando maior sua preferência na seleção para a próxima geração.

Nesta etapa a representação dos indivíduos se deu por um único gene de valores reais. A implementação do algoritmo se encontra no arquivo

`multi-objective_problem/nsgaii.py`

E a fronteira das soluções não dominadas obtida para o problema é:



**9 - Medição e interpretação dos resultados para problemas mono-objetivo**

Para os problemas mono-objetivos com domínios binário e real utilizamos o teste de hipótese U de Wilcoxon-Mann-Whitney. Este teste não paramétrico mede tem como hipótese nula que as duas amostras testadas fazem parte de uma mesma distribuição. Isso é confirmado quando o resultado p-valor do teste é maior que $\alpha < 0.5$.

Usamos tal teste para verificar se os algoritmos sGA e cGA convergem para soluções compatíveis (mesma distribuição) quando colocados para resolver os mesmos problemas. Foi utilizado pacote scipy da biblioteca padrão do Python que conta com uma implementação do teste no módulo scipy.stats.mannwhitneyu. Os resultados para os problemas mono-objetivos estudados seguem abaixo:

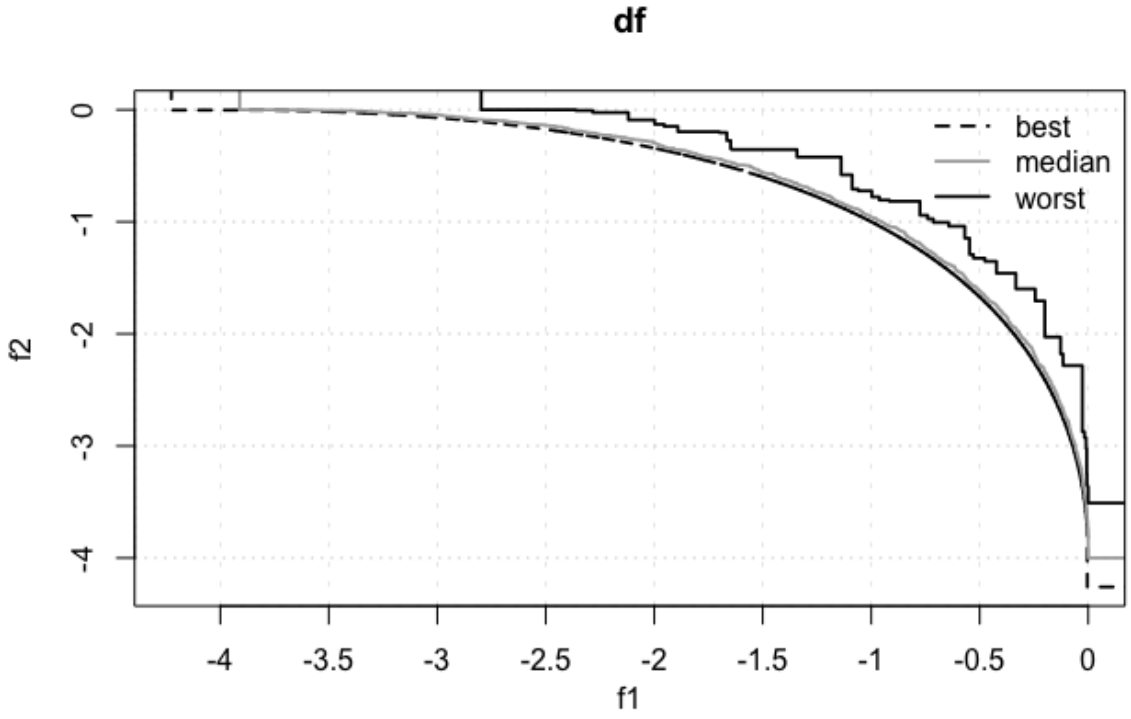
| Problema | p-valor | Interpretação |
|---|---|---|
| One-max | 0,5 | Não rejeita $H_0$: distribuições iguais |
| Trap-5 | <0,001 | Rejeita $H_0$: distribuições diferentes |
| Inv-Trap-5 | 0,282 | Não rejeita $H_0$: distribuições iguais |
| Sphere | 0,063 | Não rejeita $H_0$: distribuições iguais |
| Rosenbrock | <0,001 | Rejeita $H_0$: distribuições diferentes |

Interpretamos que foi possível obter os mesmos resultados utilizando sGA e cGA nos problemas one-max, inv-trap-5 e sphere e que o teste indicou que para trap-5 e Rosenbrock as distribuições finais não eram as mesmas. Isso pode ser explicado pelo fato destes problemas possuírem ótimos locais em que as soluções ficam aprisionadas.

**10 - Empirical Attainment Function para problema multi-objetivo**

Na seção 8 desenvolvemos o algoritmo bi objetivo NSGA-II para duas funções reais, obtendo o hiper-volume do espaço soluções não dominadas (ótimo de Pareto). Nesta seção vamos apresentar a distribuição de hiper-volumes para este problema quando executamos a solução um número grande de vezes. Chamamos este gráfico de Empirical Attainment Function e ele revela a distribuição das fronteiras não dominadas para soluções obtidas aleatoriamente.

Para isso usamos o pacote eaf da linguagem R, rodando a solução com os mesmos parâmetros 100 vezes. O resultado obtido é exibido abaixo:



## 11 - Conclusões

Comprovamos a viabilidade de uso de algoritmos genéticos para problemas de otimização numérica em problemas selecionados. Avaliados fatores como tempo de processamento e consumo de recursos, esse conjunto de técnicas pode ser uma alternativa em problemas do mundo real tais como são outras técnicas numéricas mais difundidas, como programação linear.

Através das analises gráficas concluímos a eficiência do cGA sobre o sGA no aspecto de velocidade de convergência para os valores ótimos da função objetiva, apresentando tempo de processamento muito semelhante, sendo que é conhecida sua superioridade no consumo de memória.

Para o problemas multi-objetivo, obtivemos a fronteira de soluções não dominadas conforme o esperado para o problema proposto.

## Referências

Gaspar-Cunha, A., Takahashi, R., e Antunes, C. H. (2012). Manual de Computação Evolutiva e Metaheurística. Imprensa da Universidade de Coimbra/Coimbra University Press.

Harik, G., Lobo, F. G., Goldbrg, D. E., (1999). The Compact Genetic Algorithm. IEEE Transactions on Evolutionary Computation, Vol 3, no. 4

Blank, J. Deb, K. 2020. pymoo: Multi-objective Optimization in Python. Michigan State University

Vargas, D. (2018). Um Estudo dos Parâmetros do Algoritmo NSGA-II com operador

SBX em Problemas de Otimização Estrutura Multiobjetivo. Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, Vol. 6, no. 2s