

UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE FLORESTAL
CIÊNCIA DA COMPUTAÇÃO

JOSE GRIGORIO NETO(3046)
TAÍS BATISTA DOS SANTOS (3036)



META-HEURÍSTICAS
TRABALHO PRÁTICO 3

FLORESTAL, MG
4 de dezembro de 2020

JOSE GRIGORIO NETO(3046)
TAÍS BATISTA DOS SANTOS (3036)



META-HEURÍSTICAS TRABALHO PRÁTICO 3

Documentação do 3º trabalho prático de Meta-heurísticas que tem como objetivo o desenvolvimento e a aplicação de dois algoritmo de meta-heurísticas

Orientador: Prof. Dr. Marcus Henrique Soares Mendes

FLORESTAL, MG
4 de dezembro de 2020

Sumário

1	Introdução	3
2	Desenvolvimento	4
2.1	Decisões	4
2.1.1	Parâmetros do AG	4
2.1.2	Critério de parada	5
2.2	Algoritmo Genético	5
2.3	Simulated Annealing	6
3	Resultados obtidos	9
3.1	Problema 1	9
3.2	Problema 2	9
4	Conclusão	11
5	Referências	12

1 Introdução

Este trabalho consiste na implementação e aplicação de um algoritmo genético e também o Simulated Annealing para os seguintes problemas, a fim de aplicar alguns dos conhecimentos adquiridos em aula:

$$\text{Min. } f(x) = (2\sqrt{2}x_1 + x_2) \times l$$

S.t.

$$g_1(x) = \frac{\sqrt{2}x_1 + x_2}{\sqrt{2x_1^2 + 2x_1x_2}} P - \sigma \leq 0$$

$$g_2(x) = \frac{x_2}{\sqrt{2x_1^2 + 2x_1x_2}} P - \sigma \leq 0$$

$$g_3(x) = \frac{1}{\sqrt{2x_2 + x_1}} P - \sigma \leq 0$$

$$0 \leq x_i \leq 1, \quad i = 1, 2$$

$$l = 100 \text{ cm}, P = 2 \text{ kN/cm}^2, \sigma = 2 \text{ kN/cm}^2$$

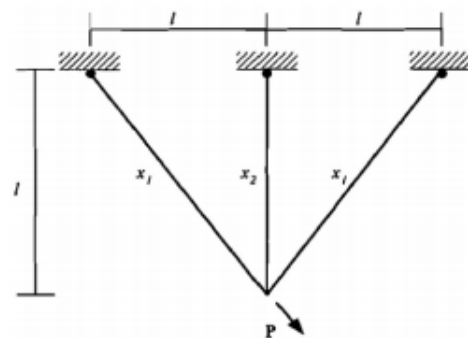


Figura 1 – Problema 1

$$\text{Min. } f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

S.t.

$$g_1(x) = -x_1 + 0.0193x_3 \leq 0$$

$$g_2(x) = -x_2 + 0.00954x_3 \leq 0$$

$$g_3(x) = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1,296,000 \leq 0$$

$$g_4(x) = x_4 - 240 \leq 0$$

$$0 \leq x_i \leq 100, \quad i = 1, 2$$

$$10 \leq x_i \leq 200, \quad i = 3, 4$$

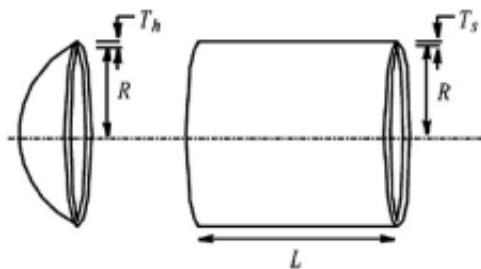


Figura 2 – Problema 2

2 Desenvolvimento

2.1 Decisões

Nesta sessão apresentaremos algumas decisões importantes para o projeto dos algoritmos.

2.1.1 Parâmetros do AG

Aqui serão apresentadas as constantes utilizadas como parâmetros nos algoritmos genéticos para os problemas 1 e 2:

- N: Tamanho da população;
- T: Quantidade de indivíduos da elite, utilizamos 10% da população;
- mutation: Chance de uma variável não se mutar durante a criação de um indivíduo;
- maxGen: O número máximo de gerações que serão executadas por aquele algoritmo;
- select_i: Número de indivíduos que serão selecionados aleatoriamente para o torneio.

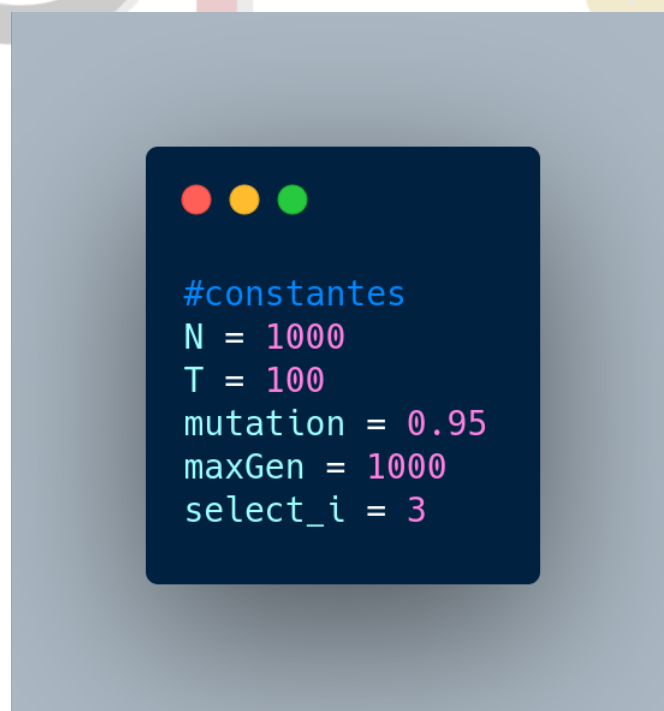


Figura 3 – Constantes do AG no problema 1

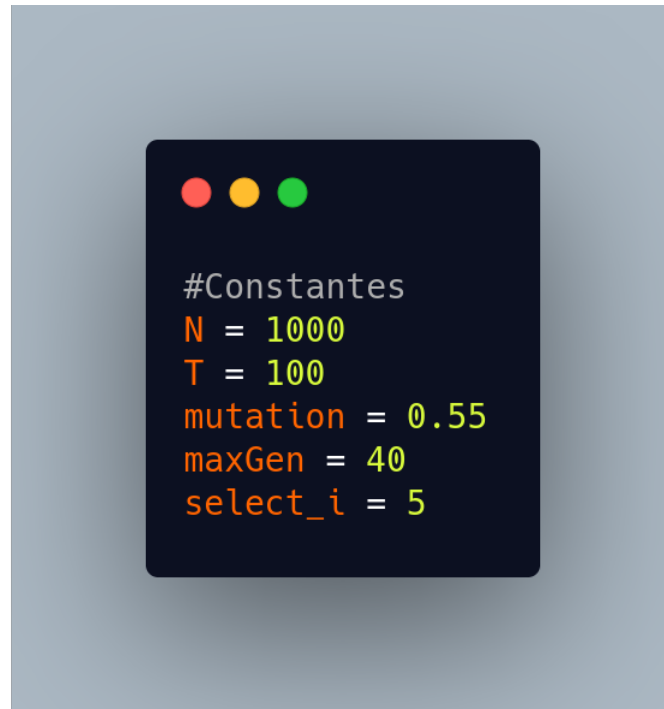


Figura 4 – Constantes do AG no problema 2

No problema 1, o número de maxGen é grande pois um dos critérios de parada é justamente esperar a solução melhorar menos que 0.0000001, portanto gostaríamos que o algoritmo executasse um número grande de gerações, porém para o problema 2 não observamos nenhuma melhora em usar um critério parecido, por isso o número máximo de gerações é bem reduzido.

2.1.2 Critério de parada


O critério de parada definido para o algoritmo SA foi o de quando a temperatura, iniciando de 90 chega a ser menor ou igual a 0.1.

Para os algoritmos genéticos, como os parâmetros para cada problema foram modificados para melhorar o desempenho de cada um, para o primeiro problema, o critério foi o de atingir 1000 gerações ou até quando a solução melhorasse menos que 0.0000001. No segundo o problema o critério de parada era atingir 40 gerações, pois foi observado que não havia uma grande melhora ao aumentar esse parâmetro.

2.2 Algoritmo Genético

O nosso Algoritmo Genético foi desenvolvido da seguinte forma: Sua população inicial é inicializada aleatoriamente dentro da restrição de tamanho de cada variável, enquanto a população é gerada, o valor de fitness de cada solução é calculado, adicionando um certo peso para cada função de restrição que não é cumprida. A seleção do algoritmo

é por torneio, sendo um torneio de tamanho 3 para o primeiro problema e um de tamanho 5 para o segundo, e seu cruzamento é discreto, para o problema 2, e aritmético para o problema 1. E por fim, a função de mutação é chamada por todos os filhos durante a sua criação, nela, cada variável do indivíduo possui uma probabilidade de se mutar, para isso, foi utilizada uma modificação da mutação uniforme. Abaixo podemos ver a função de mutação para o problema 1:



```
def mutate(self):  
    for i in range(2):  
        if(rd.uniform(0, 1) > mutation):  
            u = rd.uniform(0, 1)  
            pert = 0.1 * ((2*u) - 1)  
            self.x[i] = self.x[i] + pert
```

Figura 5 – Função de mutação do problema 1


2.3 Simulated Annealing

Para o Simulated Annealing, utilizamos a mesma função para os dois problemas, mudando apenas as funções *g* e *f* de cada problema. A função recebe os vetores de limites inferiores e superiores das variáveis de decisão e a função objetivo por parâmetro, inicia o vetor de variáveis de decisão com valores aleatórios dentro dos limites de cada variável e enquanto a temperatura atual é maior que 0.1, ele continua buscando algum vizinho cuja solução seja melhor que a atual, dependendo da temperatura é possível aceitar uma solução pior para procurar por novos vizinhos que possam melhorar a solução.

```
def simulated_annealing(lwrLim, uprLim, quality = f):  
    initial_temp = 90  
    current_temp = initial_temp  
    current_state = init(lwrLim, uprLim)  
    solution = current_state  
  
    while current_temp > final_temp:  
        newSolution = adjust(solution, lwrLim, uprLim)  
        cost_diff = quality(solution) - quality(newSolution)  
  
        if cost_diff > 0:  
            solution = newSolution  
        else:  
            if rd.uniform(0, 1) < math.exp(cost_diff / current_temp):  
                solution = newSolution  
            current_temp = current_temp / (1 + (beta * current_temp))  
    return solution
```

Figura 6 – Função HC

No caso do problema 2, assim como no algoritmo genético, foram utilizadas variáveis inteiras para x_1 e x_2 , a função de checar restrições também foi modificada para adicionar pesos maiores para este problema, como podemos ver na comparação abaixo:



```
#problema 1
def check_restritions(x):
    x1 = x[0]
    x2 = x[1]
    peso = 0
    if(not (g1(x) <= 0 )):
        peso = peso + 100000
    if(not (g2(x) <= 0 )):
        peso = peso + 100000
    if(not (g3(x) <= 0 )):
        peso = peso + 100000
    return peso

#problema 2
def check_restritions(x):
    peso = 1
    if(not (g1(x) <= 0 )):
        peso = peso + 10000000
    if(not (g2(x) <= 0 )):
        peso = peso + 10000000
    if(not (g3(x) <= 0 )):
        peso = peso + 10000000
    if(not (g4(x) <= 0 )):
        peso = peso + 10000000
    return peso
```

Figura 7 – Função de checagem de restrições

3 Resultados obtidos

3.1 Problema 1

Algoritmo	Mínimo	Máximo	Média	desvio padrão
AG	263.895843	263.896750	263.895971	0.000193
SA	263.926936	276.761199	266.895444	3.211625

Problema 1 com $0 \leq x_1, x_2 \leq 1$

Parâmetro	x_1	x_2
Valor	0.788677509	0.4082415746

Parâmetro	g_1	g_2	g_3	f
Valor	-1.0313614406e-09	-1.46410925	-0.535890750	263.895843

Melhor solução encontrada do problema 1 usando AG

Parâmetro	x_1	x_2
Valor	0.7898521810	0.405230026

Parâmetro	g_1	g_2	g_3	f
Valor	-0.000229065	-1.467651573	-0.532577491	263.926936

Melhor solução encontrada do problema 1 usando SA

Podemos ver que todos valores encontrados para o AG são menores. O desvio padrão dos algoritmos é bem discrepante. Os dados observados, nos dizem que os algoritmos funcionaram da forma que era esperada.

3.2 Problema 2

Algoritmo	Mínimo	Máximo	Média	desvio padrão
AG	6061.246092	6444.964622	6135.529031	82.209252
SA	6081.804618	11348.876718	8341.667088	985.383496

Problema 2 com limites $\{0 \leq x_1, x_2 \leq 100 \text{ e } 10 \leq x_3, x_4 \leq 200\}$

Parâmetro	x ₁		x ₂	x ₃		x ₄	
Valor	0.8125(13x0.0625)		0.4375(7x0.0625)	42.09279079358011		176.74274080536026	

Parâmetro	g ₁	g ₂	g ₃	g ₄	f
Valor	-0.000307530	-0.036032841	-110.081022507	-63.146067302	263.895843

Melhor solução encontrada do problema 2 usando AG

Parâmetro	x ₁		x ₂		x ₃		x ₄	
Valor	0.8125(13x0.0625)		0.4375(7x0.0625)		41.92734517861851		178.81748015383357	

Parâmetro	g ₁		g ₂		g ₃		g ₄		f	
Valor	-0.003302238		-0.0375131269		-271.045131034		-61.182519846		263.926936	

Melhor solução encontrada do problema 2 usando SA

Podemos ver que assim como no problema 1, todos valores encontrados para o AG são menores. O desvio padrão dos algoritmos também é discrepante. Os dados observados, nos dizem que os algoritmos funcionaram da forma que era esperada.



4 Conclusão

Ao longo deste trabalho foi possível observar o funcionamento de um algoritmo genético em comparação com o do simulated annealing. Ficou evidente que o desempenho do AG é melhor que o do SA, mesmo que sua execução demore mais, e ficamos contentes com os resultados dos algoritmos. Além disso, desenvolver um algoritmo genético foi uma experiência incrível, aplicar todos os conceitos estudados em aula serviu como um reforço da matéria, que é um assunto muito discutido hoje em dia.



5 Referências

Aula 11-14 - Algoritmos genéticos. Disponível em: https://www2.cead.ufv.br/sistemas/pvanet/Conteudo/lista_topicos.php Acesso em: 04 de dezembro de 2020.

ASKARZADEH, A. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. Computers and Structures, v. 169, p. 1–12, June 2016.

