NOVA IMS
Information Management School

ALGORITMOS GENÉTICOS

MÉTODOS COMPUTACIONAIS PARA A OPTIMIZAÇÃO

Prof. Leonardo Vanneschi
Prof. Nuno Rodrigues

Alice Vale R20181074
Eva Ferrer R20181110
Helena Oliveira R20181121
Raquel Sousa R20181102

Novembro 2020

DESCRIÇÃO DO ALGORITMO

Import & Função de Fitness

Na primeira célula de código procedemos ao *import* das várias *libraries* necessárias para a implementação do projeto. De seguida definimos a função *Rastrigin* com os limites mínimo e máximo de -5.12 e 5.12 tanto para a variável x como para y e visualizámos o seu gráfico tridimensional.

Definimos uma classe para guardar os vários indivíduos, com 2 funções. Uma recebe os valores correspondentes às coordenadas de um individuo (x,y) e a outra calcula o valor do fitness respetivo.

Criação da População

De seguida definimos a função que nos permite criar a população. Para tal fizemos 3 ciclos: o primeiro para garantir que um quarto da população é gerado através de uma distribuição uniforme de intervalos $x \in [4,5.12] \wedge y \in [-4,-5.12]$; o segundo quarto da população tem também uma distribuição uniforme com parâmetros $x \in [-4,-5.12] \wedge y \in [4,5.12]$; os restantes indivíduos, metade da população, são gerados aleatoriamente através de uma distribuição uniforme no espectro da função [-5.12,5.12]. No fim, ordenámos o *array population* que contém os indivíduos de forma ascendente com base no fitness de cada solução.

Torneio

Na célula seguinte procedemos à criação de uma função denominada *tournament* que faz a seleção dos indivíduos com base em Torneios de tamanho 5, que são posteriormente guardados no *array selected*. Para tal selecionámos 5 indivíduos aleatórios com o uso da função *random.choices()*, para garantir que a escolha é feita com reposição, ou seja, cada indivíduo pode ser selecionado mais do que uma vez. Criámos um *array tournament_selection* para guardar os 5 indivíduos escolhidos, que depois ordenámos de forma ascendente pelo valor de fitness. Depois selecionámos a primeira solução do *array tournament_selection*, que corresponde ao indivíduo com o menor fitness do grupo, e juntámos ao *array selected*. Isto é feito até o tamanho deste *array* ser igual ao da população.

Operadores Genéticos

Os indivíduos que ficaram no *array selected* vão ser sujeitos a um dos operadores genéticos. Para a mutação gerámos um valor aleatório inteiro que pode tomar o valor 0 ou 1, com 50% de probabilidade, através da função *random.randint().*

Assim, a mutação ocorre em x ou y de maneira equiprovável. Para tal asseguramos uma perturbação de baixa dimensão, proveniente de uma distribuição normal de média 0 e desvio padrão 0.5, que provém de uma análise comparativa com vários outros valores de desvio-padrão. É importante salientar que, como se trata de uma distribuição normal, quanto menor o valor do desvio-padrão maior a probabilidade de obter um valor de perturbação baixo.

Testámos com um desvio-padrão de 0.25 e obtivemos uma má performance devido ao número elevado de gerações necessárias para atingir o mínimo (≈85 gerações), sendo que em vários *runs* esse valor nem sequer foi atingido. Este comportamento deve-se ao facto de esta perturbação ser demasiado baixa para o domínio da nossa função, explorando pouco o espaço de pesquisa. O valor $\sigma = 0.75$ foi também alvo de testes, tendo sido excluído por, apesar de apresentar a melhor performance testada (≈27 gerações), a diferença do número de runs necessárias não ser significativa quando comparada com o valor obtido ao definir um desvio-padrão de 0.5 (≈30 gerações). Para além disto, como as diretrizes do enunciado requerem uma perturbação baixa, o $\sigma = 0.5$ apresenta-se como um bom compromisso entre boa performance e valor de perturbação adequada.

A figura abaixo ilustra os possíveis valores para as perturbações mencionadas. Para o caso escolhido de STD 0.5, existe 90% de probabilidade de a perturbação estar compreendida no intervalo $[-0.823;\ 0.823]$.

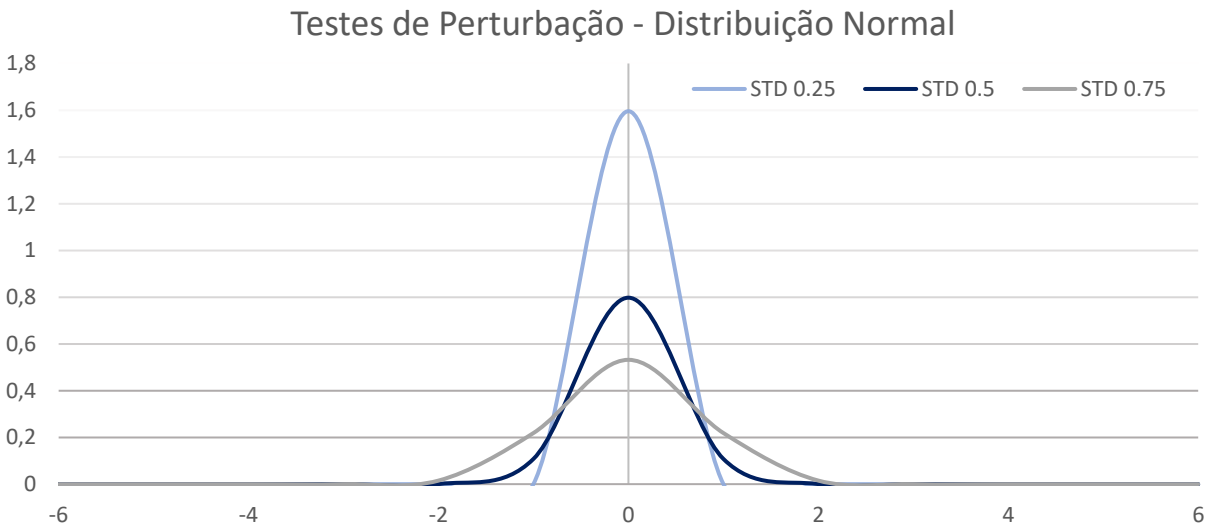

Figura 1 : Testes de perturbação

Para o crossover tivemos de gerar 2 valores inteiros aleatórios que tomam os valores 0 ou 1 porque a função vai receber dois indivíduos. Sendo que r1 está associado ao indivíduo 1 e r2 ao indivíduo 2, o parâmetro referente a cada individuo escolhe qual a coordenada que vai sofrer alterações e o outro escolhe a coordenada que o vai substituir. Exemplificando para o primeiro individuo, se r1 for igual a 0, o x vai ser alvo de crossover e será substituído por uma das coordenadas do segundo indivíduo, com base no valor de r2, formando o novo indivíduo. Para o segundo indivíduo o processo é semelhante. No final vão ser retornados os dois novos indivíduos.

Na seguinte célula procedemos à aplicação dos operadores com uma probabilidade de escolha do algoritmo genético 70/30 a favor do crossover. Tal vai ser controlado com o uso de uma variável aleatória decimal r, criada com recurso à função *random.random()*, que toma qualquer valor entre 0 e 1. Assim, se este valor for menor ou igual a 0.3 a mutação é aplicada, senão é aplicado crossover. De salientar que quando a população apenas tem 1 espaço disponível, o algoritmo vai ser forçado a utilizar a mutação. Para controlar este acontecimento e deixar 3 espaços reservados para o elitismo, o processo de escolha dos operadores genéticos só acontece enquanto existirem mais de 5 espaços disponíveis na população. No fim, o elitismo vai proceder à cópia integral dos 3 melhores indivíduos da população anterior (*parents*).

Implementação do algoritmo

Na célula final procedemos à implementação do algoritmo criado, usando as funções anteriormente descritas para uma população de tamanho 100 e durante 100 gerações (ou até atingir o critério de paragem). Para cada geração o algoritmo vai guardar os menores valores de fitness até chegar a 0 (mínimo global) ou às 100 gerações. Este ciclo é criado 30 vezes para populações diferentes e independentes, ficando a solução ótima de cada *run* guardada no *array best_values*, para posteriores análises estatísticas.

COMPARAÇÃO COM *GA_STANDARD*

Alterações

- Função de fitness *Rastrigin* e ajuste dos seus limites;
- Ajuste dos parâmetros da distribuição que gera a população;
- Aplicação do Torneio de tamanho 5;
- Definição do tamanho da população (n=100), número de gerações (gens=100);
- Aplicação de 30 testes ao algoritmo.

Valores obtidos

```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 1.9899349078238586,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 1.989923575759022,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 1.989919699346352,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 1.9899478931171544]
```

Figura 2: Vetor de soluções do GA_Standard.

```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]
```

Figura 3: Vetor de soluções do GA_desenvolvido.

	GA_Standard	GA_desenvolvido
Média	0.265	0
Mediana	0	0

Tabela 1: Estatísticas Descritivas

Pela observação dos valores acima podemos concluir que o algoritmo desenvolvido para o projeto é mais eficiente do que o standard, mesmo com as alterações efetuadas. Como estamos perante um problema de minimização e a média do nosso algoritmo corresponde ao mínimo global, significa que em todos os runs chegámos ao valor 0, o que não aconteceu no *GA_Standard*.

Assim podemos concluir que a pior performance do *GA_Standard* pode ser explicada pelos seguintes aspetos:

- Perturbação de menor dimensão, o que resulta numa menor exploração da *landscape*, e que tende a gerar valores de fitness parecidos com os dos pontos iniciais, tornando a mutação infrutífera.
- A probabilidade de escolha de operadores genéticos é igual (50/50), o que não é de todo desejável visto que a mutação vai causar possíveis saltos aleatórios e descontrolados nos pontos, já que faz *Exploration*. O crossover, por se basear em *Exploitation*, vai causar movimentos mais controlados, dado que não se vai afastar muito dos pontos iniciais. Assim, o nosso algoritmo apresenta-se mais adequado a estes movimentos, porque tem uma menor probabilidade de ocorrer mutação.
- Ausência de elitismo de tamanho 3, que se mostra uma grande desvantagem devido ao facto de os algoritmos genéticos tradicionais perderem os melhores indivíduos com as transformações. Assim, o algoritmo desenvolvido salvaguarda pelo menos o top 3 dos melhores valores de fitness.

No entanto, é de salientar um ponto a favor do algoritmo *GA_Standard*:

- A completa aleatoriedade da seleção dos indivíduos da população, gerados através de uma distribuição uniforme no espectro da função, vai garantir uma maior diversidade de pontos, quando comparada com o algoritmo desenvolvido. Assim, ao definirmos metade da população em intervalos de pontos específicos e simétricos, e sendo a função também simétrica, geramos pontos com valores de fitness muito parecidos pelo que a diversidade de pontos não é tão vasta.

SIMULATED ANNEALING VS GENETIC ALGORITHM

O Simulated Annealing (SA) aplicado a este problema teria uma pior performance quando comparado com o Algoritmo Genético (GA). Tal deve-se:

- À maior probabilidade de terminar retornando um valor de ótimo local dada a elevada rugosidade da landscape de fitness, porque apesar de conseguir piorar soluções isso é mais provável de acontecer no início do algoritmo, devido ao elevado valor do parâmetro de controlo. Este vai diminuindo ao longo do algoritmo e por isso vai aumentar a chance de terminar num mínimo local e não ótimo.
- Devido ao parâmetro de controlo existente, a fase de exploração deste algoritmo é breve e mais provável de ocorrer inicialmente enquanto que no GA pode ocorrer durante toda a implementação. Tal vai provocar uma menor versatilidade do SA pois a partir de um dado ponto vai ficar restringido apenas à *Exploitation*.
- O GA vai precisar de menos iterações para chegar a uma solução ótima dado que analisa um conjunto de indivíduos para o cálculo da solução, enquanto o SA analisa apenas um indivíduo em cada iteração.

BIBLIOGRAFIA

- Vanneschi, Leonardo, Slides das aulas teóricas de Métodos Computacionais para a Otimização, 2020