

Swarm Intelligence - Seminário
Inteligência Computacional
2021/2022
Projeto - Fase II

Conteúdo

1. Computação Evolucionária	3
Algoritmos genéticos	3
2. Inteligência Swarm	3
Definição de Swarm	3
Algoritmo PSO	4
.....	5
Algoritmo ACO	5
3. Cuckoo Search Optimization(CSO).....	6
Inspiração natural.....	6
Voos de Lévy	6
Parâmetros	7
Pseudocódigo	7
Análise do código	7
.....	8
Comparação CSO vs PSO	8
4. Análise de desempenho.....	9
Funções “benchmarks”	9
Testes realizados	9
• Função Ackley:	9
• Função Esfera Dimensão 2:	10
• Função Esfera Dimensão 3	10
PSO vs CSO.....	10
• Função Ackley:	10
• Função Esfera Dimensão 2:	11
• Função Esfera Dimensão 3	11
5. Conclusão	12
6. Trabalho futuro	12
Referências	13

1. Computação Evolucionária

A computação evolucionária é um ramo da inteligência computacional que surgiu nos anos 50 e que se pode descrever como um processo de otimização global inspirado na Teoria de Evolução de Darwin.

Sistemas de computação evolucionária resolvem problemas a partir de vários parâmetros tais como populações, erro e acerto, meta-heurística, ou otimização estocástica.

Algoritmos genéticos

Grande parte dos algoritmos de computação evolucionária pertencem a um sistema de algoritmos, os algoritmos evolucionários.

Estes algoritmos podem ser implementados apartir dos seguintes passos:

- Gere uma população inicial (e.g. aleatoriamente).
- Repita os passos até o término:
 - Selecione os indivíduos mais aptos para reprodução (e.g. maior fitness);
 - Gere novos indivíduos a partir dos selecionados (e.g. via crossover e mutação).

Dentro destes sistemas os algoritmos mais conhecidos são os algoritmos genéticos.

Estes algoritmos foram criados a meio da década de 70 por John Holland e têm por bases técnicas inspiradas na biologia evolutiva tais como hereditariedade, mutação, seleção natural e recombinação.

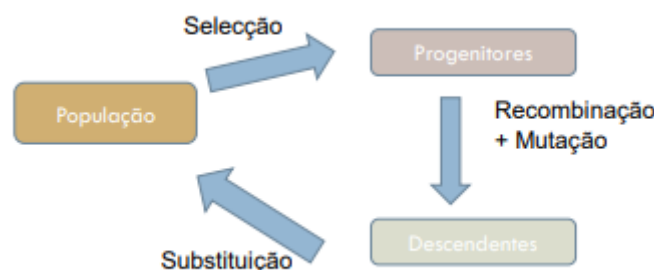


Fig.1 – Funcionamento dos algoritmos genéticos

2. Inteligência Swarm

A computação evolucionária não é apenas composta por algoritmos evolucionários tal como o algoritmo genético, mas também é composta por algoritmos de otimização meta-heurística. Dentro destes mesmos algoritmos podemos encontrar algoritmos de inteligência Swarm.

Definição de Swarm

A inteligência Swarm é aquela encontrada no comportamento de enxame (swarm).

Um "swarm" define-se como um conjunto estruturado de indivíduos (ou agentes) que interagem entre si para atingirem um objetivo comum, de forma mais eficiente do que agindo individualmente. Os indivíduos são relativamente simples, contudo o

comportamento coletivo pode ser complexo – analogia com colónias de formigas, enxames, bandos de pássaros, cardumes, etc.

Algoritmo PSO

O principal algoritmo da inteligência swarm é o PSO – Particle Swarm Optimization.

Este algoritmo é baseado na modelação do comportamento social de aves em bandos.

Um exemplo deste comportamento é as aves quando se encontram em bando para minimizar a perda de energia tendem a se alinhar em V, sendo que o posicionamento de cada ave é obtido a partir do comportamento da ave anterior.

Apartir da perceção destes mesmos padrões elaborou-se o algoritmo PSO.

O algoritmo funciona de acordo com os seguintes princípios:

- Cada partícula representa uma possível solução para um problema de otimização;
- A posição de uma partícula é determinada iterativamente de acordo com a sua inércia, “experiência individual” e “experiência das partículas vizinhas” (Figura 2).

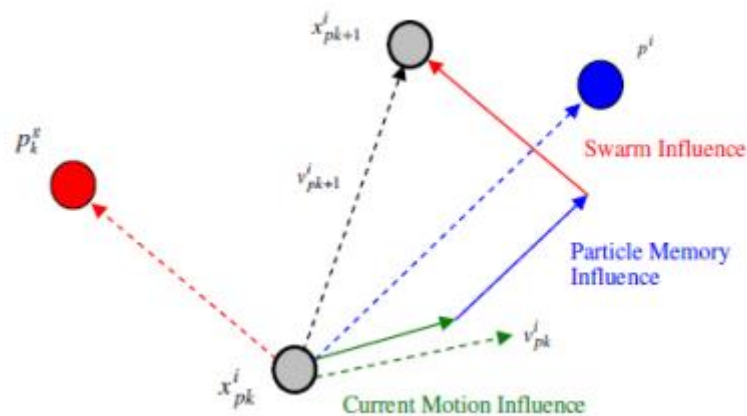


Fig.2-Cálculo da posição de uma partícula

Para o algoritmo PSO estão definidas várias topologias de como os vários indivíduos interagem. Na seguinte imagem estão representadas estas mesmas topologias:

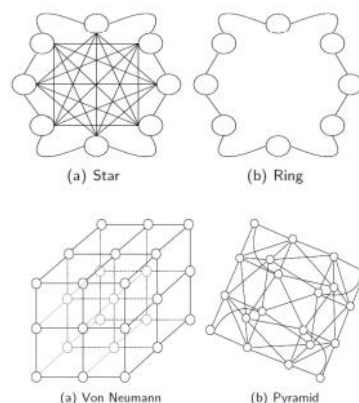


Fig. 3 – Topologias do algoritmo Swarm

1. Inicializar o *swarm* de partículas $P(t)$ para $t=0$, de tal forma que a posição $P_i \in P(t)$, de cada partícula $x_i(t)$ é aleatória, dentro do espaço multidimensional de procura.
2. Avaliar a performance de cada partícula na sua posição atual $F(x_i(t))$
3. Comparar a performance de cada indivíduo com a sua melhor performance:
Se $F(x_i(t)) < pbest_i$ Então:
(a) $pbest_i = F(x_i(t))$
(b) $xbest_i = x_i(t)$
4. Comparar a performance de cada indivíduo com melhor performance global:
Se $F(x_i(t)) < gbest$ Então:
(a) $gbest = F(x_i(t))$
(b) $xgbest = x_i(t)$
5. Atualiza a velocidade de cada partícula:
$$v_i(t) = v_i(t-1) + \rho_1(xbest_i - x_i(t)) + \rho_2(xgbest - x_i(t))$$

onde ρ_1 e ρ_2 representam valores aleatórios positivos, respetivamente designados de componente cognitiva e componente social.
6. Move cada partícula para a sua nova posição:
(a) $x_i(t) = x_i(t-1) + v_i(t)$
(b) $t = t + 1$
7. Volta ao passo 2 e repete até convergência do algoritmo.

Fig.4- Algoritmo PSO versão global best através da tipologia em estrela

Algoritmo ACO

O algoritmo designado de ACO considera ‘swarms’ com um número elevado de indivíduos, podendo existir indivíduos com diferentes características e comportamento (ao contrário do PSO, onde todos os indivíduos são considerados semelhantes). Apesar das eventuais diferenças, todos os indivíduos contribuem para um objetivo comum. Considere um grupo de formigas cujo objetivo é de encontrar alimento. Cada formiga possui um comportamento muito limitado de perceção e comunicação (neste caso indireta). Enquanto se deslocam no espaço de pesquisa, de forma mais ou menos aleatória, elas libertam feromonas. Uma formiga encontra possui uma maior probabilidade de mover-se para uma região com maior densidade de feromonas. Depois de um tempo, a maioria das formigas escolherá o mesmo caminho (Figura 1).

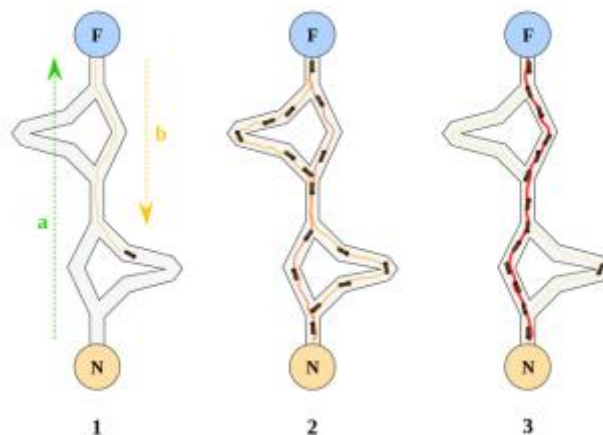


Fig.5-demonstração do uso das feromonas

3. Cuckoo Search Optimization(CSO)

Outro algoritmo meta-heurístico é o cuckoo search optimization(CSO) desenvolvido por Xin-She Yang and Suash Deb em 2009 e é inspirado no método em como os cucos garantem a continuidade da sua espécie.

Inspiração natural

O Cuco é uma ave parasita, isto é, ao invés de construir um ninho para colocar os seus ovos metem em ninhos de outras aves hospedeiras. (sendo que algumas espécies podem-se envolver diretamente.)

Certas espécies de cuco retiram os ovos da espécie onde colocaram os seus ovos para aumentar a probabilidade de sobrevivência dos seus. As fêmeas de cuco também são especializadas no mimetismo na reprodução dos ovos para estes serem semelhantes aos dos hospedeiros. Usualmente, os cucos também inserem os seus ovos em ninhos onde as aves hospedeiras acabaram de pousar os seus ovos.

Quando os pássaros hospedeiros reparam que os ovos não são seus estes retiram os ovos do Cuco do seu ninho ou então largam o ninho e constroem outro.

O CSO é baseado em 3 regras:

- Cada cuco põe um ovo que representa um conjunto de soluções coordenadas, que em determinado momento é largado em um ninho aleatório
- Uma fração dos ninhos contendo os melhores ovos ou soluções vão ser selecionados para as próximas gerações
- O número de ninhos é fixo e há uma probabilidade dos hospedeiros encontrarem um ovo de cuco(pa). Se isto acontecer, o hospedeiro pode mandar o ovo de cuco fora ou contruir um novo ninho num local diferente.

Voos de Lévy

Os desenvolvedores deste algoritmo descobriram que o melhor método matemático para simular a aleatoriedade com que o cuco escolhe o ninho onde vai largar o ovo (descrita na primeira regra).

Os voos de Lévy são um método do tipo Random walk em que os passos são definidos em respeito á distancia entre estes, sendo que a direção tem uma probabilidade aleatória. O próximo movimento é baseado na posição atual.

Sendo a representação deste mesmo algoritmo a seguinte:

$$X_{i,t+1} = X_{i,t} + \alpha \oplus \text{Levy}(\lambda)$$

em que $\alpha > 0$ é o tamanho do passo. O tamanho aleatório do passo é seguido por uma distribuição de Lévy $\rightarrow \text{Levy} \sim u = t^{-\lambda} \quad 1 < \lambda \leq 3$

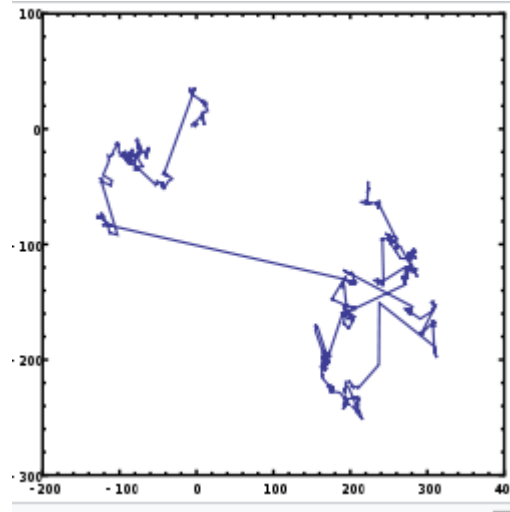


Fig. 5 – Representação de um voo de Lévy

Parâmetros

Os parâmetros deste algoritmo são os seguintes p_a , que é a probabilidade dos hospedeiros encontrarem o ovo e n , que é o número de ninhos de hospedeiros disponíveis

Pseudocódigo

```
Objective function:  $f(\mathbf{x})$ ,  $\mathbf{x} = (x_1, x_2, \dots, x_d)$ ;
Generate an initial population of  $n$  host nests;
While ( $t < \text{MaxGeneration}$ ) or (stop criterion)
    Get a cuckoo randomly (say,  $i$ ) and replace a solution by its solution by performing Lévy flights;
    Evaluate its quality/fitness  $F_i$ 
    [For maximization,  $F_i \propto f(\mathbf{x}_i)$  ];
    Choose a nest among  $n$  (say,  $j$ ) randomly;
    if ( $F_i > F_j$ ),
        Replace  $j$  by the new solution;
    end if
    A fraction ( $p_a$ ) of the worse nests are abandoned and new ones are built;
    Keep the best solutions/nests;
    Rank the solutions/nests and find the current best;
    Pass the current best solutions to the next generation;
end while
```

Análise do código

1. Geração de um número inicial de ninhos:

```
super(cso, self).__init__()
self.__Nests = []

beta = 3 / 2
sigma = (gamma(1 + beta) * sin(pi * beta / 2) / (
    gamma((1 + beta) / 2) * beta *
    2 ** ((beta - 1) / 2))) ** (1 / beta)
u = np.array([normalvariate(0, 1) for k in range(dimension)]) * sigma
v = np.array([normalvariate(0, 1) for k in range(dimension)])
step = u / abs(v) ** (1 / beta)

self.__agents = np.random.uniform(lb, ub, (n, dimension))
self.__nests = np.random.uniform(lb, ub, (nest, dimension))
Pbest = self.__nests[np.array([function(x)
    for x in self.__nests]).argmin()]

Gbest = Pbest
self.__points(self.__agents)
```


2. Entra em um ciclo percorrendo todos os ninhos:

```
for t in range(iteration):
    a. Apanha um cuco aleatoriamente e reformular a sua solução:
    for i in self.__agents:
        val = randint(0, nest - 1)
        if function(i) < function(self.__nests[val]):
            self.__nests[val] = i

    b. Avalia a sua qualidade com a função objetiva;
    fnests = [(function(self.__nests[i]), i) for i in range(nest)]
    fnests.sort()
    fcuckoos = [(function(self.__agents[i]), i) for i in range(n)]
    fcuckoos.sort(reverse=True)

    nworst = nest // 2
    worst_nests = [fnests[-i - 1][1] for i in range(nworst)]

    c. Escolhe um ninho aleatoriamente;
    for i in worst_nests:
        if random() < pa:
            self.__nests[i] = np.random.uniform(lb, ub, (1, dimension))

    d. Se a solução nova for melhor que a anterior:
        i. A solução "anterior" passa a ser a solução nova.
    if nest > n:
        mworst = n
    else:
        mworst = nest

    e. Uma porção dos piores ninhos é abandonada e são criados novos ninhos;
    for i in range(mworst):
        if fnests[i][0] < fcuckoos[i][0]:
            self.__agents[fcuckoos[i][1]] = self.__nests[fnests[i][1]]

    f. Mantém as melhores soluções/ninhos;
    self.__nests = np.clip(self.__nests, lb, ub)
    self.__Levyfly(step, Pbest, n, dimension)
    self.__agents = np.clip(self.__agents, lb, ub)
    self.__points(self.__agents)
    self.__nest()

    g. Faz um ranking das melhores soluções e avalia qual é a melhor solução atualmente;
    Pbest = self.__nests[np.array([function(x)
                                   for x in self.__nests]).argmin()]

    if function(Pbest) < function(Gbest):
        Gbest = Pbest

    h. Passa as melhores soluções para a próxima geração.
    self.__set_Gbest(Gbest)
```

Comparação CSO vs PSO

As grandes vantagens do CSO face ao PSO é que este mais fácil de aplicar e apresenta menor número de parâmetros, w , $c1$ e $c2$. E além disso em alguma da bibliografia é sugerido que este algoritmo apresenta melhor desempenho face aos outros métodos meta-heurísticos, apesar de ser um algoritmo que tende a encontrar soluções ótimas locais.

Esta característica anteriormente referida pode ser uma desvantagem quando pretendemos encontrar uma solução ótima global. Outra desvantagem comparativamente é uma taxa de convergência ser lenta.

4. Análise de desempenho

Funções “benchmarks”

Para analisar o desempenho de algoritmo deste tipo existem funções “benchmark” que são funções de teste matemáticas que normalmente apresentam um mínimo global e vários locais, podendo ser um dos objetivos dos algoritmos que usam estas funções aproximar-se do mínimo global.

Nesta análise de desempenho utilizamos a função esfera e a função ackley para avaliar o nosso algoritmo.

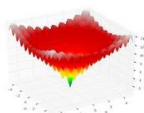
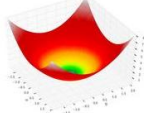
Name	Plot	Formula	Global minimum	Search domain
Ackley function		$f(x, y) = -20 \exp \left[-0.2 \sqrt{0.5 (x^2 + y^2)} \right] - \exp[0.5 (\cos 2\pi x + \cos 2\pi y)] + e + 20$	$f(0, 0) = 0$	$-5 \leq x, y \leq 5$
Sphere function		$f(x) = \sum_{i=1}^n x_i^2$	$f(x_1, \dots, x_n) = f(0, \dots, 0) = 0$	$-\infty \leq x_i \leq \infty, 1 \leq i \leq n$

Fig.6 – Funções benchmark utilizadas

Testes realizados

Quanto aos parâmetros globais que são abrangidos pela maior parte dos algoritmos de inteligência coletiva colocamos os parâmetros default isto é:

Parametros Globais ¶

```

]: ...
    Global parameters:
        n: number of agents
        function: test function
        lb: lower limits for plot axes
        ub: upper limits for plot axes
        dimension: space dimension
        iteration: number of iterations
    ...
n=100
lb= -10
ub= 10
dimension= 2
iteration= 100

```

Nesta fase de testes dividimos o teste em duas partes. Na primeira parte testamos a alteração dos dois parâmetros base do algoritmo. Sendo que para isso fixamos o valor de um dos parâmetros e fomos alterando o valor do outro.

- **Função Ackley:**

nest=100 Pa=variável

Pa	0.1	0.25	0.5	0.75
Gbest	-0.058, -0.1455	0.047, 0.62	-0.02, -0.36	-0.0001, 5.989e-5

nest=variável Pa=0.5

Nest	200	400	1000	10000
Gbest	0.00517, 0.0007	0.085, -0.098	-7.421e-6, 0.004	2.733e-8, 0.028

- **Função Esfera Dimensão 2:**

nest=100 Pa=variável

Pa	0.1	0.25	0.5	0.75
Gbest	0.12, -0.272	0.245, -0.045	-0.033, -0.096	-0.034, -0.035

nest=variável Pa=0.5

Nest	200	400	1000	10000
Gbest	0.075, -0.043	0.0046, -0.0021	0.01062, 0.076	-0.044, 0.0021

- **Função Esfera Dimensão 3**

nest=100 Pa=variável

Pa	0.1	0.25	0.5	0.75
Gbest	0.418, -0.047, 0.02	0.313, -0.0001, -0.053	0.178, -0.226, -0.009	-5.5e-15, -7.02e-16, -3.35e-15

nest=variável Pa=0.5

Nest	200	400	1000	10000
Gbest	-3.71e-11, 2.46e-10, 1.05e-9	-0.195, -0.018, 0.09	-0.157, 0.343, 0.069	-0.00089, 0.0572, -0.0220

PSO vs CSO

Na segunda fase comparamos os melhores resultados entre o nosso algoritmo e a versão standard do PSO:

- **Função Ackley:**

CSO Pa=0.5 n=10000
2.733e-8, 0.028

PSO standard
9.351e-14, 1.54e-13

- Função Esfera Dimensão 2:

CSO Pa=0.5 n=400
0.0046, -0.0021

PSO standard
-3.1495e-14, 1.39e-14

- Função Esfera Dimensão 3

CSO Pa=0.75 n=400
-5.5e-15, -7e-16, -3.35e-15

PSO standard
-2.21e-12, 2.14e-12, -1.5e-12

Todos estes resultados encontram-se representados no jupyter notebook enviado junto a este relatório assim como a representação gráfica destes mesmos resultados.

Conclusão e trabalho futuro (1 página) Apresentar nesta secção as principais conclusões e o trabalho a realizar para a fase III.

5. Conclusão

Com este trabalho prático podemos analisar um algoritmo de pesquisa com base no comportamento de parasitagem dos cucos em relação aos seus ovos.

A partir dos testes realizados anteriormente conseguimos observar mais detalhadamente as funcionalidades do algoritmo, tanto através dos resultados obtidos bem como da análise da representação gráfica.

Quanto aos parâmetros do próprio CSO ao analisarmos alguma bibliografia até dos próprios autores é sugerido o uso do parametro $p_a=0.25$ e $nest = O(L/10)$ ou $nest=O(L/10)$ em que L é escala característica do problema em análise.

Apesar disso podemos analisar que esta sugestão não se aplica às funções de teste analisadas e testadas assim pois:

Analisando os resultados podemos concluir que face a tendência tanto quanto usando a função Ackley como usando a função esfera existe uma tendência de quanto maior o parâmetro, melhor o global best obtido, tanto para o parâmetro como para o parâmetro nest.

Quando comparado ao PSO obtivemos resultados piores sendo que este facto se deve maioritariamente à tendência do CSO cair em local bests.

6. Trabalho futuro

Na próxima meta pretendemos “traduzir” o código elaborado na meta 1 de matlab para python bem como utilizar o CSO, através do estudo e análise de resultados obtidas nesta meta para treinar os pesos dos Hiper parâmetros ou até mesmo a partir deste mesmo algoritmo para seleccionar este mesmos Hiper parâmetros.

Referências

Material disponibilizado ao longo das aulas da unidade curricular

https://www.matec-conferences.org/articles/matecconf/pdf/2018/91/matecconf_eitce2018_03003.pdf

<https://www.ijarcce.com/upload/2016/november-16/IJARCCE%20119.pdf>

https://www.researchgate.net/publication/235799431_Automated_Test_Data_Generation_Using_Cuckoo_Search_and_Tabu_Search_CSTS_Algorithm?fbclid=IwAR08ej7-owlF7H5VKKzIBBY0emjvKJyh2I-XqebeJ49KDF-Eolt1r1cFf6E

https://www.researchgate.net/publication/272853718_Comparative_Study_of_Krill_Herd_Firefly_and_Cuckoo_Search_Algorithms_for_Unimodal_and_Multimodal_Optimization?fbclid=IwAR1oioy_covrMaTfd_wqgTNSbX0vkQAiyq7kP_EppW0_YMy18SDZLAW-NqI

https://www.youtube.com/watch?v=46wezNBhKA&t=964s&ab_channel=RitikaxRayPixy

https://www.youtube.com/watch?v=BmZq1hi9eQg&ab_channel=Decoderz