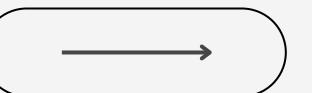


IDENTIFICAÇÃO DA SAÚDE DE PRODUTOS VEGETAIS

Fase 2



A202110042

JORGE RICARDO MARQUES DUARTE

Introdução



A identificação precoce e eficaz de doenças em plantas emerge como uma solução vital no âmbito do desenvolvimento sustentável.

Introdução à Computação Swarm



Paradigma

É um paradigma que imita o comportamento de animais sociais. Centra-se em como padrões complexos surgem de interações simples, privilegiando a descentralização e a auto-organização.

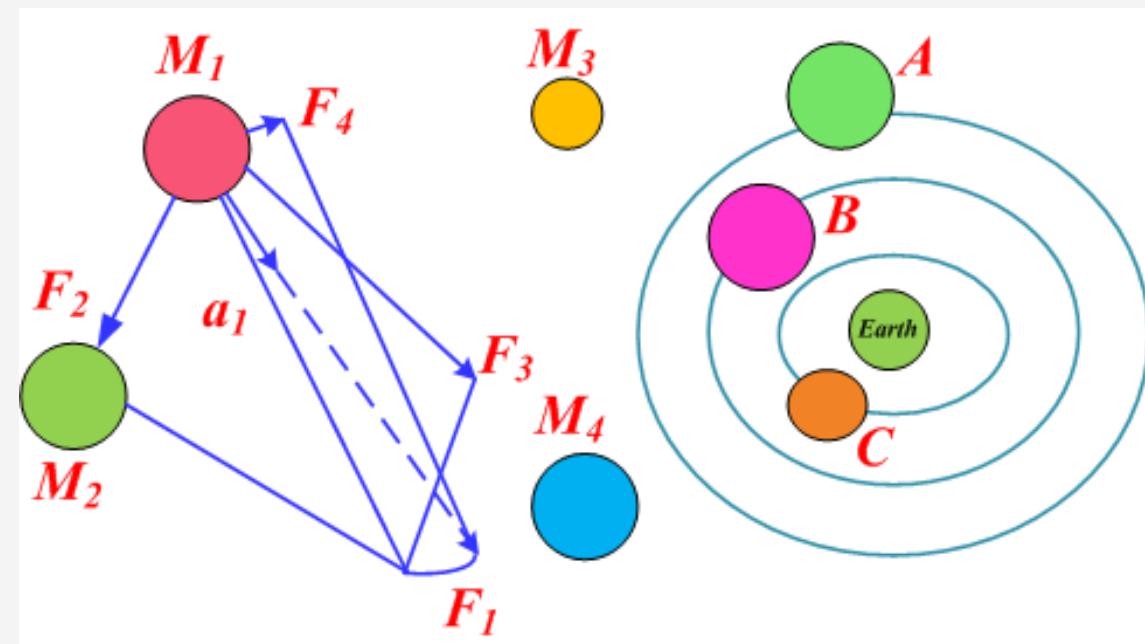
Inteligência Coletiva

A inteligência emerge da colaboração de vários agentes autónomos. Cada agente contribui para um objetivo comum, seguindo regras básicas e respondendo a informações locais, sem necessidade de controlo central.

Aplicações

Usada em áreas como otimização, robótica e inteligência artificial.

Gravitational Search Algorithm (GSA)



Inspiração

Inspirado nas leis da gravidade e na dinâmica dos corpos celestes no espaço. Baseia-se na ideia de que cada massa (ou agente) atrai outras massas, criando um sistema em que as partículas se movem influenciadas pela força gravitacional.

Mecanismo

Cada agente possui uma massa que é proporcional à qualidade da solução que representa. As forças gravitacionais entre os agentes direcionam a pesquisa para soluções mais promissoras, com massas maiores atraindo massas menores.

Aplicação

Usado para encontrar soluções ótimas em problemas de otimização complexos.

Efetivo na exploração do espaço de soluções, especialmente útil em problemas com vastos e complexos espaços de pesquisa.

Comparação entre GSA e PSO

PSO

Mais partículas melhoraram a exploração do espaço.

Aumentar as iterações foi crucial, especialmente para evitar convergência prematura.

Pode enfrentar desafios ao aumentar a dimensão do problema.

GSA

Mostrou melhor desempenho na minimização da função de Ackley, com resultados mais próximos de zero.

Eficaz em espaços de busca extensos e complexos.

Benchmarks

```
class AckleyBenchmark:
    @staticmethod
    def ackley_2d(x):
        return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x[0]**2 + x[1]**2))) - np.exp(0.5 * (np.cos(2 * np.pi * x[0]) + np.cos(2 * np.pi * x[1]))) + np.exp(1) + 20

    @staticmethod
    def ackley_3d(x):
        if len(x) == 2:
            return AckleyBenchmark.ackley_2d(x)
        return -20 * np.exp(-0.2 * np.sqrt(1/3 * (x[0]**2 + x[1]**2 + x[2]**2))) - np.exp(1/3 * (np.cos(2 * np.pi * x[0]) + np.cos(2 * np.pi * x[1]) + np.cos(2 * np.pi * x[2]))) + np.exp(1) + 20

def run_gsa_benchmark(dimensions, max_iter, num_agents, benchmark_function):
    gsa_obj = gsa(n=num_agents, function=benchmark_function, lb=-5, ub=5, dimension=dimensions, iteration=max_iter, G0=3)
    best_cost = gsa_obj.get_Gbest()
    best_position = gsa_obj.get_agents()
    return best_cost, best_position

def run_pso_benchmark(dimensions, max_iter, num_particles, benchmark_function):
    pso_obj = pso(n=num_particles, function=benchmark_function, lb=-5, ub=5, dimension=dimensions, iteration=max_iter)
    best_cost = pso_obj.get_Gbest()
    best_position = pso_obj.get_agents()
    return best_cost, best_position

dimensions_list = [2, 3]
max_iter_list = [50, 100, 200, 300]
num_agents_particles_list = [10, 20, 30]
```

	Algorithm	Dimensions	Max Iter	Num Agents/Particles	Best Cost
5	GSA	2	100	20	[0.032725238681468795, 0.01542941881455527]
6	GSA	2	100	30	[-0.007917361778434488, -0.015447984545644132]
7	GSA	2	200	10	[0.031184736360074183, 0.025498999368368885]
8	GSA	2	200	20	[0.01228125219911845, 0.009153127322616528]
9	GSA	2	200	30	[0.013031935698197165, 0.005101065501063581]
10	GSA	2	300	10	[-0.0016859833069990497, -0.0005560086746243314]
11	GSA	2	300	20	[-0.001098493512591333, 0.0011273204004384925]
12	GSA	2	300	30	[0.00386968252747312, 0.0023661515842293237]
13	GSA	3	50	10	[0.24037238176040038, -0.09548616505740581, -0.14333875946268249]
14	GSA	3	50	20	[-0.05824842066464436, 0.009977817648523624, -0.04708075371773185]
15	GSA	3	50	30	[-0.007841918063225706, -0.04151903041705286, 0.08772712951589628]
16	GSA	3	100	10	[-0.0809634259139935, 0.058475808979977395, 0.01647800277539721]
17	GSA	3	100	20	[-0.050939634365499406, -0.05049238451295532, -0.01853486257240433]
18	GSA	3	100	30	[0.001330607848367013, 0.00915278915368739, -0.0031750360638406333]
19	GSA	3	200	10	[0.010511452524960635, 0.008333522285012232, -0.006209929433255823]
20	GSA	3	200	20	[-0.002018119984972341, -0.00697819812277331, 0.011911238005358232]
21	GSA	3	200	30	[0.004342030585954081, 0.007558609998530218, -0.00210274609586003...]
22	GSA	3	300	10	[-0.014106702513068024, -0.00498267072466203, -0.003205190501825517]
23	GSA	3	300	20	[-0.005665589216118914, 0.015770452263366295, -0.004444282311814301]
24	GSA	3	300	30	[-0.002185561990790681, -0.003574607080327967, 0.002133370880843594]
25	PSO	2	50	10	[-1.3072198606236606e-06, -6.057067033233053e-06]
26	PSO	2	50	20	[-3.705270045499179e-06, -1.0744405495593677e-08]
27	PSO	2	50	30	[4.1152862938225404e-07, -1.3672070116235454e-07]
28	PSO	2	100	10	[1.9260657707918362e-11, 2.345291760015715e-11]
29	PSO	2	100	20	[2.076110679099389e-12, -1.941541294082919e-12]
30	PSO	2	100	30	[-6.924344424917844e-14, 9.34452105196454e-13]
31	PSO	2	200	10	[-1.297250742445252e-16, 1.9176376734954874e-16]
32	PSO	2	200	20	[-1.2903034252067907e-15, 8.326286248710565e-16]
33	PSO	2	200	30	[2.0121097558268409e-16, -2.2721870815015323e-16]
34	PSO	2	300	10	[-5.4705108699770673e-17, -2.5592584607138223e-16]
35	PSO	2	300	20	[1.7952379633059105e-15, -7.341635122000183e-16]
36	PSO	2	300	30	[-1.5312661323575828e-15, 2.4279720795244765e-16]
37	PSO	3	50	10	[6.356843349668398e-06, 0.9392011802337009, 4.499392020095746e-06]
38	PSO	3	50	20	[-1.319012953031255e-06, -2.7717991581714618e-05, 2.10412770548432...]
39	PSO	3	50	30	[2.7419123352948152e-06, 2.5280461074604503e-06, 6.82301579709522...]
40	PSO	3	100	10	[0.9597032345569037, 2.2758748776082066e-09, 0.9597032289002141]
41	PSO	3	100	20	[4.4453705974674075e-11, -8.774964744153926e-11, -4.85792684096968...]
42	PSO	3	100	30	[-1.7174484800581817e-11, -3.80846556331093e-11, -4.1006804231405785]

Otimização de Hiperparâmetros

Variações nas métricas de acerto e AUC indicam sensibilidade à inicialização e escolha de hiperparâmetros.

O GSA é mais lento devido à complexidade do cálculo da interação entre agentes, e foi possível observar melhorias consistentes com aumento de agentes e iterações, indicando eficácia na exploração do espaço de soluções.

O PSO é mais rápido, mas pode não explorar completamente o espaço de soluções com menos iterações ou agentes, e pode necessitar de ajustes em configurações específicas.

Função de Avaliação

```
def evaluate_model_GSA(solution):
    num_neurons, learning_rate = solution
    model = create_cnn_model(num_neurons, learning_rate)
    model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=0)

    # Predições e cálculo do AUC
    y_pred = model.predict(x_validation)
    y_validation_categorical = to_categorical(y_validation, num_classes=len(categories))
    auc_score = roc_auc_score(y_validation_categorical, y_pred, multi_class='ovr')

    return 1 - auc_score # Minimizar 1 - AUC
```

Resultados

Tipos	Num_Agentes	Num_Iterações	Dimensão	Limite_Inferior	Limite_Superior	Num_Neurônios	Learning_Rate	Train_Accuracy	Validation_Accuracy	Test_Accuracy	Test_AUC
PSO	1	3	2	[10, 0.0001]	[100, 0.01]	47.87874676243591	0.0087753845747881	0.87833330154419	0.7909374833106995	0.8025000095367432	
PSO	2	3	2	[10, 0.0001]	[100, 0.01]	75.22757411441034	0.0054160404778483	0.9545833468437196	0.8712499737739563	0.893750011920929	
PSO	2	5	2	[10, 0.0001]	[100, 0.01]	52.05586527933356	0.0014469380436244	0.9762499928474426	0.9259374737739564	0.9375	
PSO	3	3	2	[10, 0.0001]	[100, 0.01]	81.06893465593836	0.0014025599233438	0.9768333435058594	0.9293749928474426	0.9412500262260436	
PSO	3	5	2	[10, 0.0001]	[100, 0.01]	17.307716508221	0.0059809363961371	0.9321666955947876	0.8500000238418579	0.870000047683716	
PSO	3	10	2	[10, 0.0001]	[100, 0.01]	32.82104476338705	0.0018843320254153	0.9607499837875366	0.931249976158142	0.9387500286102296	
PSO	4	10	2	[10, 0.0001]	[100, 0.01]	47.96320103316841	0.0015117143188046	0.9670833349227904	0.9075000286102296	0.9162499904632568	
GSA	2	3	2	[10, 0.0001]	[100, 0.01]	47.27899332675588	0.0087431398830898	0.8589166402816772	0.7337499856948853	0.7350000143051147	0.9557892857142857
GSA	2	5	2	[10, 0.0001]	[100, 0.01]	77.86912753604364	0.007785604028644	0.897333239555359	0.801562488079071	0.8087499737739563	0.9733446428571428
GSA	3	3	2	[10, 0.0001]	[100, 0.01]	21.971680107185872	0.0023550929124363	0.9631666541099548	0.8315625190734863	0.8212500214576721	0.9871321428571428
GSA	3	5	2	[10, 0.0001]	[100, 0.01]	28.13555613472179	0.0023181631727126	0.965666651725769	0.8987500071525574	0.9024999737739564	0.9948517857142856
GSA	3	10	2	[10, 0.0001]	[100, 0.01]	70.78772267244659	0.0011889281157621	0.97200002861023	0.9334375262260436	0.9325000047683716	0.9974214285714286
CUSTOM	-	-	-	-	-	100.0	0.0015117143188046	0.9823333621025084	0.9337499737739564	0.9412500262260436	

Conclusão e Resultados

Variações nas métricas de acerto e AUC indicam a importância da escolha cuidadosa dos hiperparâmetros.

Mudança para CNNs aumentou significativamente a taxa de acerto, destacando a eficácia em classificação de imagens.

GSA e PSO mostraram-se eficientes na descoberta automática de hiperparâmetros ótimos, economizando tempo e esforço.

Q/A