Universidade Federal de Pernambuco
Centro de Informática
Programa de Pós-Graduação em Ciências da Computação

Relatório Exercício Computacional

Relatório do exercício computacional do Curso IN1131-COMPUTAÇÃO-EVOLUCIONÁRIA da Universidade Federal de Pernambuco.

Aluno: Pedro R. X. do Carmo e Luiz Felipe de B. J. Costa

Professor orientador: Aluizio Fausto Ribeiro Araujo

Professor co-orientador: Lucas Farias

Dezembro
2021

Conteúdo

1 Resumo

O exercício computacional descrito neste relatório é composto por duas questões. Na primeira questão foram implementadas as funções Ackley (d=2), Griewank (d=2), Trid (d=5) e Colville (d=4). Todas as funções foram implementadas em python (e estão presentes no arquivo objective_functions.py - em anexo a este documento). Para resolver o problema de otimização associado a essas funções foram utilizados os seguintes algoritmos: Estratégia Evolutiva, Evolução Diferencial, Algoritmos Genéticos e Algoritmos de Estimação de Distribuição. Para avaliar as soluções foram utilizadas as médias, desvio padrão e Teste t de Student.
Na segunda questão foi implementado uma função de avaliação para o problema da mochila múltipla. Um algoritmo GA foi implementado para resolver o problema. O algoritmo implementado é capaz de resolver o problema da mochila múltipla para qualquer quantidade de objetos e mochilas, no entanto o algoritmo foi executado apenas com a configuração proposta, obtendo um valor total nos itens de 33 uva (Unidade de Valor).
Foram utilizadas duas bibliotecas de python: Pymoo [1] e Deap [2].

2 Primeira Questão

Foram implementadas com o auxílio da biblioteca Pymoo [1] as quatro funções: Ackley (d=2), Griewank (d=2), Trid (d=5) e Colville (d=4). As implementações das funções estão no arquivo objective_functions.py em anexo a este relatório. Para todas as funções, foram executados os seguintes algoritmos: Estratégia Evolutiva, Evolução Diferencial, Algoritmos genéticos e Algoritmos de Estimação de Distribuição. Para todos os algoritmos foram utilizadas como critério de parada o número de gerações == 100. Abaixo, está a configuração utilizada em cada algoritmo.

- GA:

```
pop_size=100,
sampling=Random Sampling,
selection=Tournament Selection,
crossover=Simulated Binary Crossover(prob=0.9,
    eta=3),
mutation=Polynomial Mutation(eta=5),
survival=Fitness Survival
```

- DE:

```
pop_size=100,
sampling=Latin Hypercube Sampling,
variant="DE/best/1/bin",
CR=0.5
```

- ES:

```
n_offsprings=200,
rule=1.0 / 7.0,
phi=1.0,
gamma=0.85,
sampling=Random Sampling,
survival=Fitness Survival
```

- EDA:

```
centroid=5.0*n_var,
sigma=5.0,
lambda=1000,
```

```
mu=int(lambda/4)
```

2.1 Função de Ackley

função. Foi implementada a função de Ackley, onde a=20, b=0.2, c=2π;

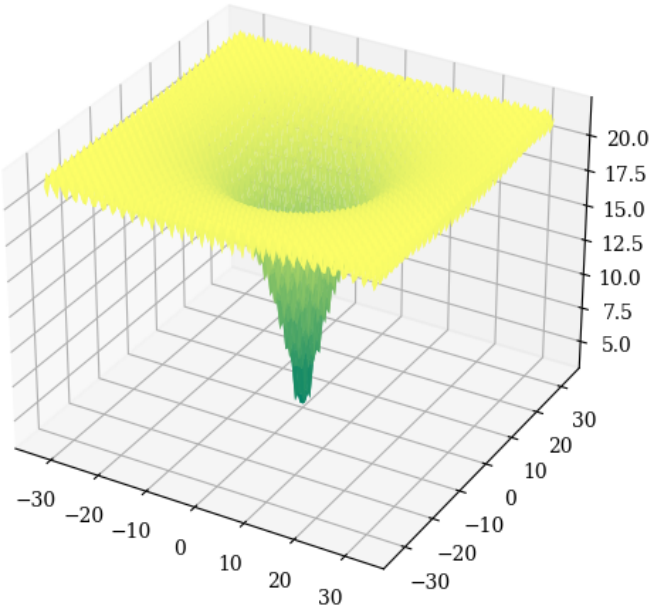
$$f(x) = -a\ exp(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - exp(\frac{1}{d}\sum_{i=1}^{d} cos(c\ x_i)) + a + exp(1)$$

$$-32.768 \leq x_i \leq 32.768$$

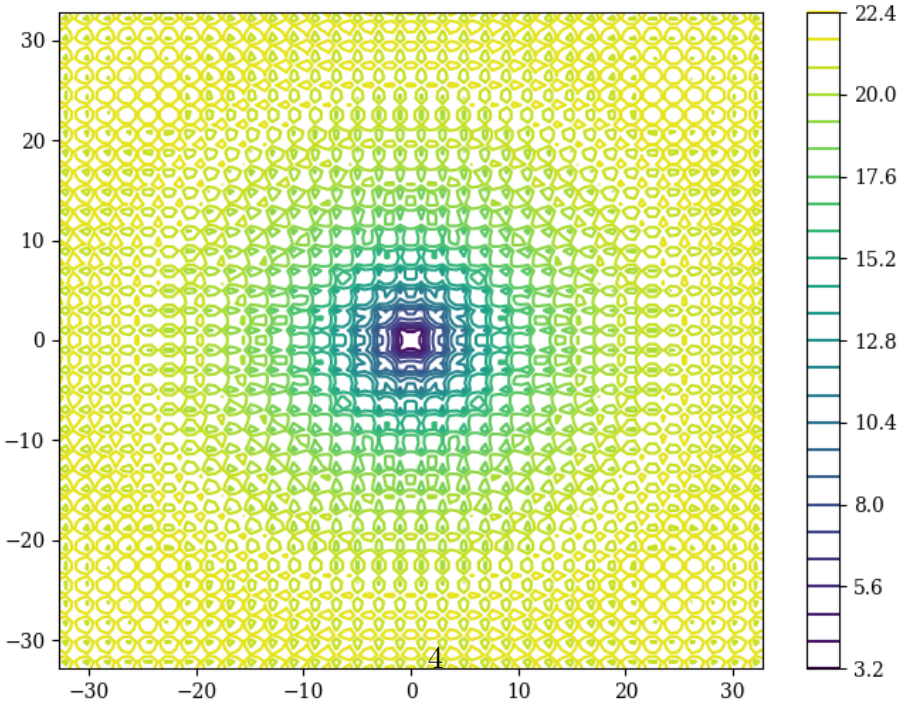
resultados. Para o algoritmo de Ackley, foram executadas 30 repetições. Os melhores resultados foram obtidos usando Evolução diferencial. O algoritmo foi capaz de alcançar uma solução que se aproxima do mínimo global. Os resultados para todos os algoritmos estão na imagem abaixo:

```
Ackley
100%|                                                                                | 30/30 [05:55<00:00, 11.84s/it]
resultados para Estrategia Evolutiva:
      solucao [-7.99920005e-15  1.84637303e-14]
      media: 8.413891805503227e-12
      desvio: 3.885408983965012e-11
resultados para Evolucao Diferencial:
      solucao [-2.23251892e-17  3.00664592e-16]
      media: 0.0
      desvio: 0.0
resultados para Algoritmos Genéticos:
      solucao [9.59021753e-04 3.91255607e-06]
      media: 0.0008313282335954853
      desvio: 0.002714736628923007
resultados para Algoritmos de Estimação de Distribuição:
      solucao [-2.19092827e-17  5.41561611e-17]
      media: 1.1842378929335005e-18
      desvio: 2.9487285734311623e-18
Teste T para os algoritmos ES e DE
    P-valor encontrado: 0.24832158991354403
    Para um nível de significância de 5% os resultados são equivalentes
Teste T para os algoritmos GA e DE
    P-valor encontrado: 0.10453619063551087
    Para um nível de significância de 5% os resultados são equivalentes
Teste T para os algoritmos EDA e DE
    P-valor encontrado: 0.034697139568251996
    O algoritmo DE obteve resultados melhores com nível de confiança 95%
O melhor algoritmo encontrado para o problema solicotado foi: DE
```

Figura 1: Resultados Ackley



(a) Fitness Landscape Ackley 1



(b) Fitness Landscape Ackley 2

2.2 Função de Griewank

função. Foi implementada a função de Griewank

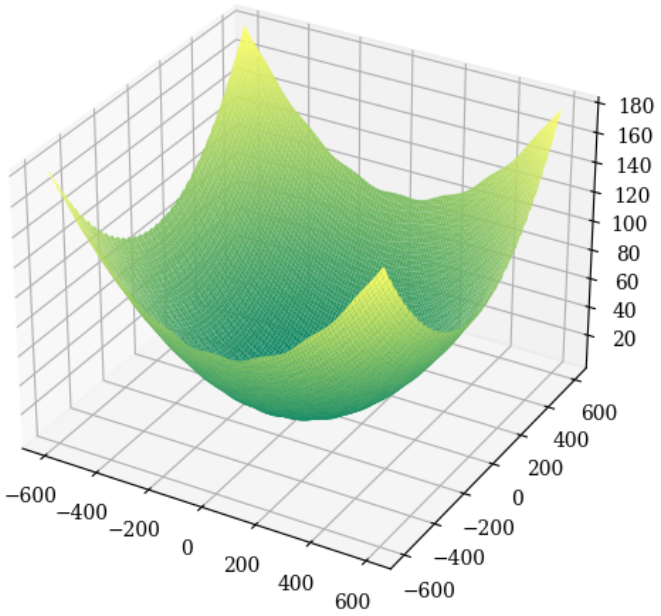
$$f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} cos(\frac{x_i}{\sqrt{i}})$$

$$-600 \leq x_i \leq 600$$

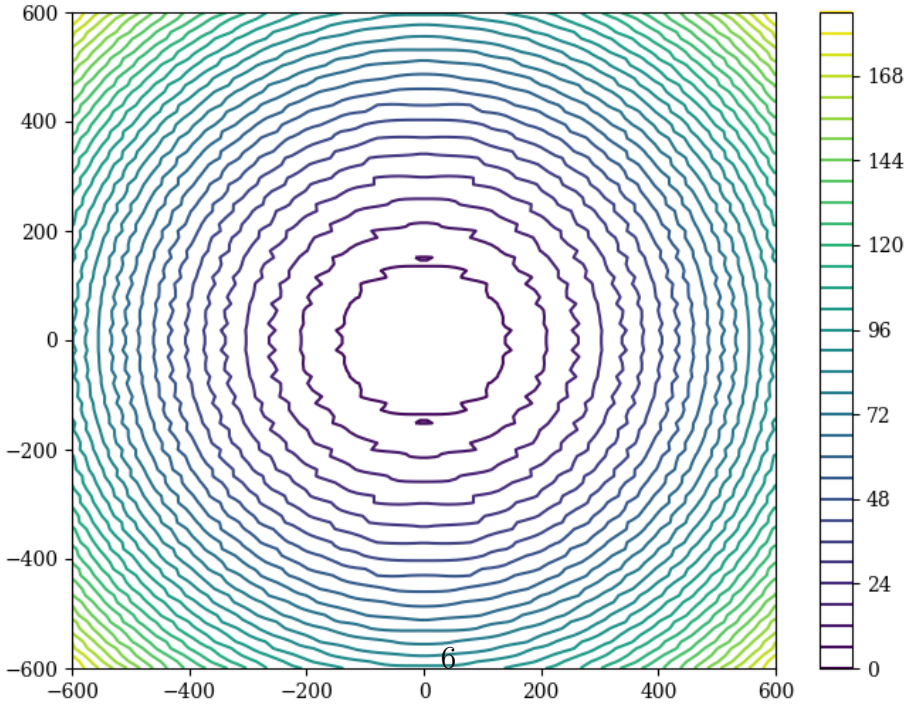
resultados. Para o algoritmo de Griewank, foram executadas 30 repetições. Os melhores resultados foram obtidos usando GA. Para essa função todos os algoritmos atingiram um mínimo local. Os resultados para todos os algoritmos estão na imagem abaixo:

```
Griewank
100%|| 30/30 [04:36<00:00,  9.23s/it]
resultados para Estrategia Evolutiva:
      solucao [-4.32334252e-09  6.77215741e-09]
      media: 0.005176620034710696
      desvio: 0.004125395436257244
resultados para Evolucao Diferencial:
      solucao [-3.14002263 -4.43844448]
      media: 0.005012981277373663
      desvio: 0.0061551200175770605
resultados para Algoritmos Genéticos:
      solucao [-6.27995921e+00 -1.14687108e-05]
      media: 0.004038977197511477
      desvio: 0.0037338191925572347
resultados para Algoritmos de Estimação de Distribuição:
      solucao [-0.00545372  0.00135499]
      media: 0.10455271730075662
      desvio: 0.29830965321829944
Teste T para os algoritmos ES e DE
    P-valor encontrado: 0.9057438927422377
    Para um nível de significância de 5% os resultados são equivalentes
Teste T para os algoritmos GA e ES
    P-valor encontrado: 0.27542851317307104
    Para um nivel de significância de 5% os resultados são equivalentes
Teste T para os algoritmos EDA e GA
    P-valor encontrado: 0.07479548523552546
    Para um nível de significância de 5% os resultados são equivalentes
O melhor algoritmo encontrado para o problema solicotado foi: GA
```

Figura 2: Resultados Griewank



(a) Fitness Landscape Griewank 1



(b) Fitness Landscape Griewank 2

2.3 Função de Colville

função. Foi implementada a função de Colville

$$f(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1),$$

resultados. Para o algoritmo de Colville, foram executadas 30 repetições. Os melhores resultados foram obtidos usando Estratégia Evolutiva. O algoritmo foi capaz de alcançar uma solução muito próxima do mínimo global. Os resultados para todos os algoritmos estão na imagem abaixo:

```
Colville
100%|| 30/30 [07:29<00:00, 14.99s/it]
resultados para Estrategia Evolutiva:
      solucao [1.00000041 1.00000082 0.99999963 0.99999922]
      media: 9.106523411718942e-13
      desvio: 1.0524117936897297e-12
resultados para Evolucao Diferencial:
      solucao [0.99103539 0.98210591 1.00928754 1.01871246]
      media: 8.615311256383198
      desvio: 23.232992899732004
resultados para Algoritmos Genéticos:
      solucao [0.56957166 0.32273709 1.30356063 1.69779213]
      media: 0.2813671883360681
      desvio: 0.320286010504641
resultados para Algoritmos de Estimação de Distribuição:
      solucao [ 1.10172099  1.22583631 -0.04977495  0.08833851]
      media: 6.625203047587234
      desvio: 2.23847921136718
Teste T para os algoritmos ES e DE
    P-valor encontrado: 0.05053147182093847
    Para um nível de significância de 5% os resultados são equivalentes
Teste T para os algoritmos GA e ES
    P-valor encontrado: 1.4754567428845065e-05
    O algoritmo ES obteve resultados melhores com nível de confiança 95%
Teste T para os algoritmos EDA e ES
    P-valor encontrado: 7.395464103432578e-23
    O algoritmo ES obteve resultados melhores com nível de confiança 95%
O melhor algoritmo encontrado para o problema solicotado foi: ES
```

Figura 3: Resultados Colville

2.4 Função de Trid

função. Foi implementada a função Trid, com $d = 5$, $-10 < x_i < 10$ e $i = 1, ..., d$

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

resultados. Para o algoritmo de Trid (d=5), foram executadas 30 repetições. Os melhores resultados foram obtidos usando Estratégia Evolutiva.

Esse algoritmo foi capaz de atingir um resultado muito próximo ao mínimo global. Os resultados para todos os algoritmos estão na imagem abaixo:

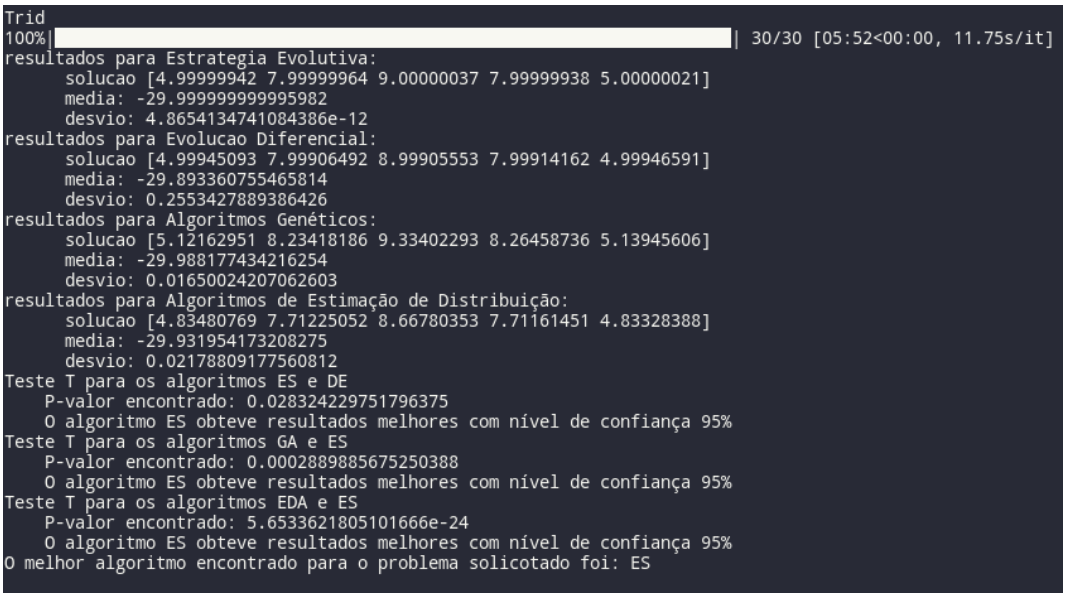


Figura 4: Resultados Trid

3 Segunda Questão

Foi implementada a função de avaliação para o problema da mochila múltipla utilizando python (classe KnapSack() do arquivo objective_functions.py do arquivo em anexo a este relatório). O problema consiste em resolver o problema de otimização onde dada uma lista de objetos com pesos e valores, e uma lista de mochilas com capacidades, deve-se dar como resposta o maior número de itens com o maior valor total a ser levado nas mochilas. Para este problema, foram levadas em conta as seguintes restrições: um objeto só pode estar em uma mochila e uma mochila não pode levar mais do que a sua capacidade definida. Para resolver este problema foi proposto um algoritmo genético com a seguinte configuração:

- KnapSack GA:

```
pop_size = num_mochilas*num_itens*20,
sampling = Random Sampling,
crossover = Exponential Crossover,
mutation = Bitflip Mutation,
selection = Random
```

O critério de parada é o número de avaliações = 50000.
Esse algoritmo recebe como entrada uma lista de tuplas (peso,valor), onde cada tupla representa um item e uma lista de mochilas com suas capacidades. Com essas informações, o algoritmo sugere quais itens levar em cada mochila de forma a maximizar o valor total levado. O algoritmo foi executado utilizando a configuração proposta no exercício: 17 objetos e 03 mochilas.
resultados. O melhor resultado encontrado para o problema foi 33 unidades de valor. Na figura abaixo, é possível observar quais itens devem ser levados em quais mochilas. O tempo total de execução do algoritmo foi de aproximadamente 9 segundos.

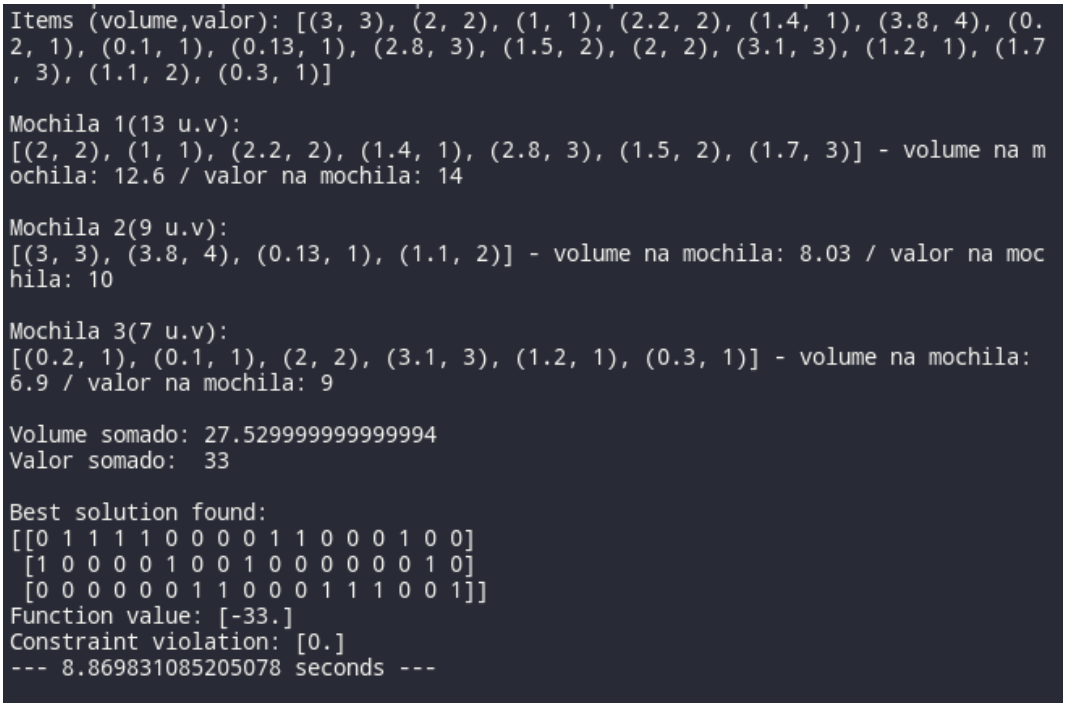


Figura 5: Resultados KnapSack O=17;M=03

Utilizando um número menor de avaliações (número de avaliações = 40000) é possível encontrar um ótimo local em tempo cerca de 30% menor. Como pode ser visto na imagem abaixo:

```
Items (volume,valor): [(3, 3), (2, 2), (1, 1), (2.2, 2), (1.4, 1), (3.8, 4), (0.
2, 1), (0.1, 1), (0.13, 1), (2.8, 3), (1.5, 2), (2, 2), (3.1, 3), (1.2, 1), (1.7
, 3), (1.1, 2), (0.3, 1)]

Mochila 1(13 u.v):
[(2, 2), (1, 1), (2.2, 2), (0.1, 1), (2.8, 3), (1.5, 2), (3.1, 3)] - volume na m
ochila: 12.7 / valor na mochila: 14

Mochila 2(9 u.v):
[(3, 3), (1.4, 1), (1.2, 1), (1.7, 3), (1.1, 2)] - volume na mochila: 8.4 / valo
r na mochila: 10

Mochila 3(7 u.v):
[(3.8, 4), (0.13, 1), (2, 2), (0.3, 1)] - volume na mochila: 6.23 / valor na moc
hila: 8

Volume somado: 27.330000000000002
Valor somado:  32

Best solution found:
[[0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 0 0]
 [1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0]
 [0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1]]
Function value: [-32.]
Constraint violation: [0.]
--- 6.592350244522095 seconds ---
```

Figura 6: Resultados KnapSack: Num_eval=40000

Referências

[1] J. Blank and K. Deb, pymoo: Multi-Objective Optimization in Python, in IEEE Access, vol. 8, pp. 89497-89509, 2020, doi: 10.1109/ACCESS.2020.2990567

[2] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau and Christian Gagné, "DEAP: Evolutionary Algorithms Made Easy", Journal of Machine Learning Research, pp. 2171-2175, no 13, jul 2012.