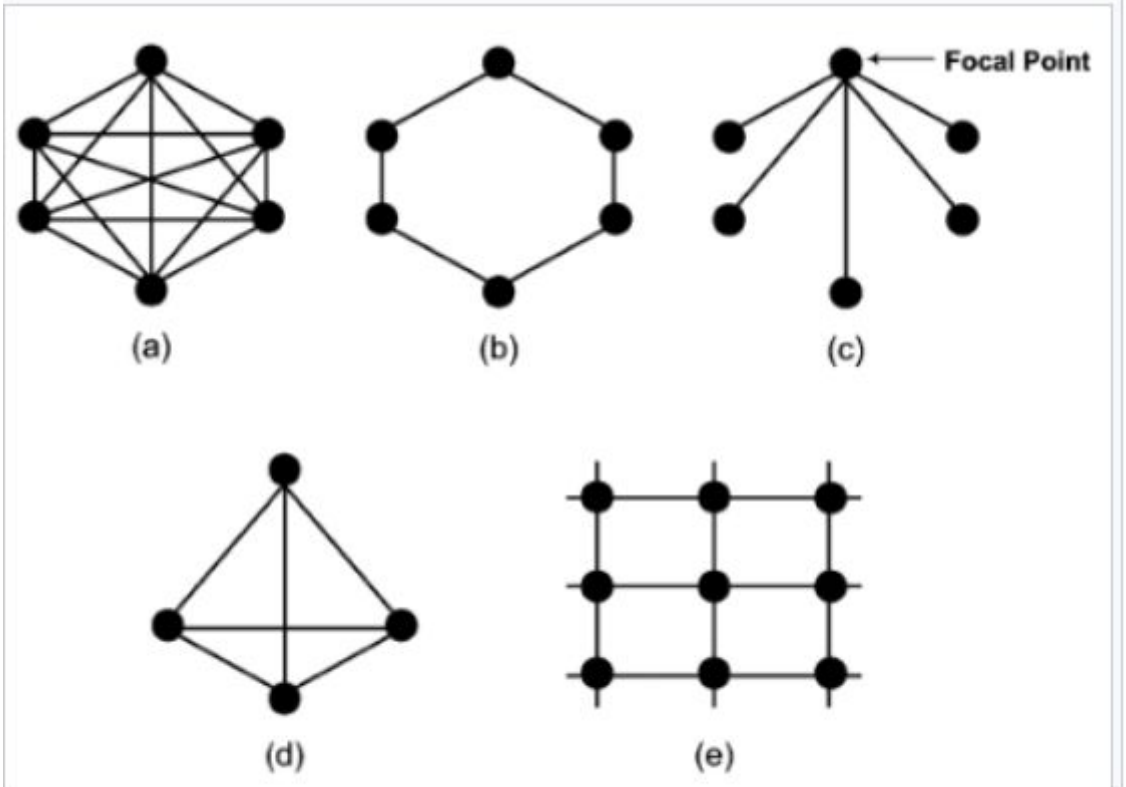
Alunos: Harã Heique e Jadson Pereira
Github: https://github.com/HaraHeique/enxame-particulas-IA

1. Como utilizar o algoritmo de otimização por enxame de partículas

Neste tutorial iremos utilizar um algoritmo de otimização por enxame de partículas para minimizar a função descrita pela função F6 de Schaffer.

1.1 Teoria

O método de enxame de partícula foi proposto por Kennedy e Eberhart em 1995 e tem como objetivo otimizar um problema iterativamente ao tentar melhorar a solução com respeito a uma dada medida de qualidade.



Swarm Typologies. (a) Melhor Global. (b) Topologia em Anel. (c)Topologia Circular. (d)Topologia em Pirâmide. (e) Topologia Von Neumann.

Figura 1 - Aplicação do algoritmo.

2. Problema proposto

Nosso problema consiste em aplicar o algoritmo PSO (“*particle swarm optimization*”) para otimizar e analisar o comportamento pessoal de partículas geradas aleatórias a partir de

domínios (d1[-100,100] , d2[-100,100]) e velocidade(v[-15,15]) pré-definidos e o número de partículas, interações e população pré-definidos pelo usuário.

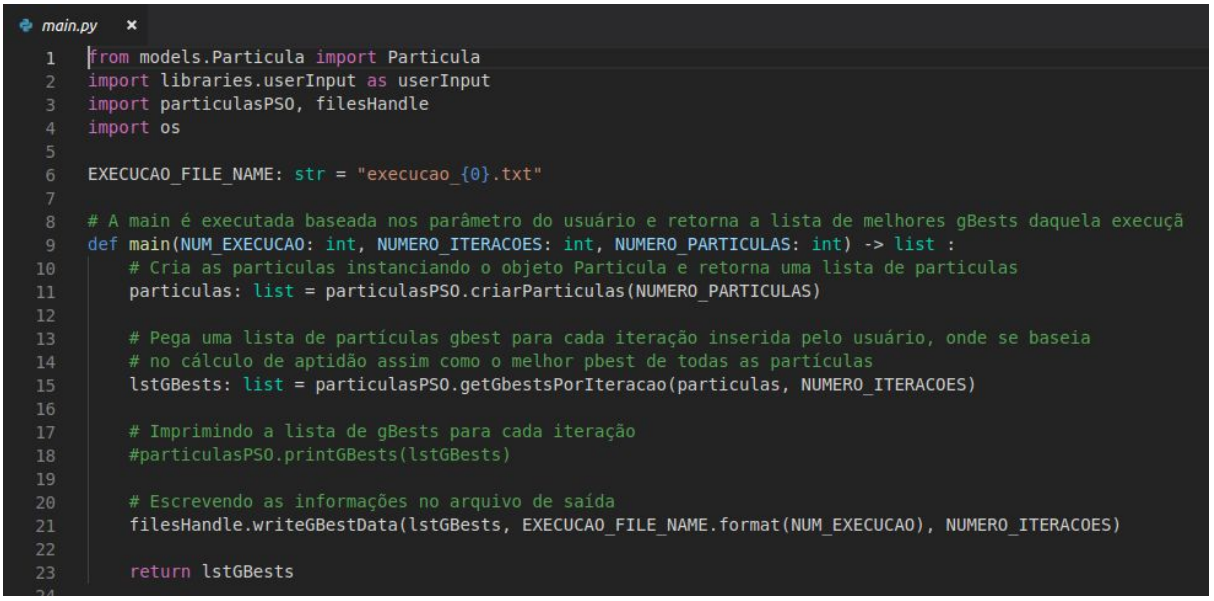
3. Implementação

Para a implementação do algoritmo de otimização utilizamos a linguagem de programação Python que vem ganhando bastante popularidade no meio da IA, devido ser uma linguagem de script bem fácil de ser aprendida e consequentemente aumento o nível de produtividade.
No nosso problema, utilizamos a estrutura de dicionário para armazenar as melhores partículas entre as interações e sua respectiva aptidão. Vale ressaltar que uma partícula é uma lista com duas posições, x0 e x1, sendo eles gerados aleatoriamente no intervalo do domínio (neste caso -100,100) .

3.1. Trechos mais importantes da implementação

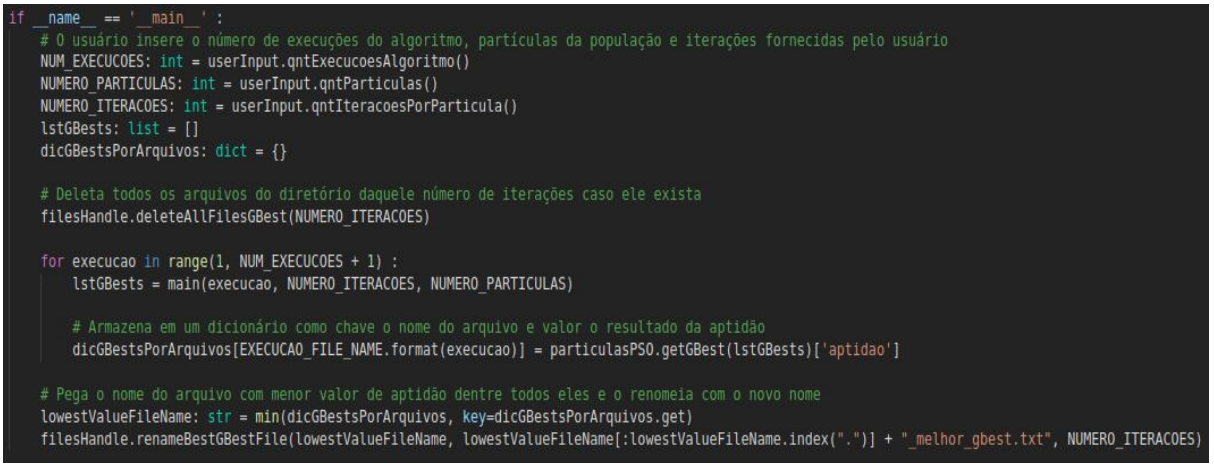
3.1.1 Partículas PSO

Primeiramente a função principal que é responsável por chamar todos os módulos e funções do algoritmo, desde a leitura dos dados que o usuário fornece (Interação, Número de população e execução), até o processamento da mesma que faz as partículas e aplica a função de otimização.



```python
from models.Particula import Particula
import libraries.userInput as userInput
import particulasPSO, filesHandle
import os

EXECUCAO_FILE_NAME: str = "execucao_{0}.txt"

# A main é executada baseada nos parâmetro do usuário e retorna a lista de melhores gBests daquela execuçã
def main(NUM_EXECUCAO: int, NUMERO_ITERACOES: int, NUMERO_PARTICULAS: int) -> list :
    # Cria as particulas instanciando o objeto Particula e retorna uma lista de particulas
    particulas: list = particulasPSO.criarParticulas(NUMERO_PARTICULAS)

    # Pega uma lista de partículas gbest para cada iteração inserida pelo usuário, onde se baseia
    # no cálculo de aptidão assim como o melhor pbest de todas as partículas
    lstGBests: list = particulasPSO.getGbestsPorIteracao(particulas, NUMERO_ITERACOES)

    # Imprimindo a lista de gBests para cada iteração
    #particulasPSO.printGBests(lstGBests)

    # Escrevendo as informações no arquivo de saída
    filesHandle.writeGBestData(lstGBests, EXECUCAO_FILE_NAME.format(NUM_EXECUCAO), NUMERO_ITERACOES)

    return lstGBests
```

Figura 2 - Main: Criação de partículas e processamento de interações.



```python
if __name__ == '__main__' :
    # O usuário insere o número de execuções do algoritmo, partículas da população e iterações fornecidas pelo usuário
    NUM_EXECUCOES: int = userInput.qntExecucoesAlgoritmo()
    NUMERO_PARTICULAS: int = userInput.qntParticulas()
    NUMERO_ITERACOES: int = userInput.qntIteracoesPorParticula()
    lstGBests: list = []
    dicGBestsPorArquivos: dict = {}

    # Deleta todos os arquivos do diretório daquele número de iterações caso ele exista
    filesHandle.deleteAllFilesGBest(NUMERO_ITERACOES)

    for execucao in range(1, NUM_EXECUCOES + 1) :
        lstGBests = main(execucao, NUMERO_ITERACOES, NUMERO_PARTICULAS)

        # Armazena em um dicionário como chave o nome do arquivo e valor o resultado da aptidão
        dicGBestsPorArquivos[EXECUCAO_FILE_NAME.format(execucao)] = particulasPSO.getGBest(lstGBests)['aptidao']

    # Pega o nome do arquivo com menor valor de aptidão dentre todos eles e o renomeia com o novo nome
    lowestValueFileName: str = min(dicGBestsPorArquivos, key=dicGBestsPorArquivos.get)
    filesHandle.renameBestGBestFile(lowestValueFileName, lowestValueFileName[:lowestValueFileName.index(".")] + "_melhor_gbest.txt", NUMERO_ITERACOES)
```

Figura 3 - Main: Entrada de dados fornecidas pelo usuário e processamento de busca do arquivo com o melhor best para renomeá-lo.

Nessa parte de código é retornado uma lista com a quantidade de partículas pré-definida pelo usuário. Vale ressaltar que os valores de posição da partícula são gerados randomicamente dentro do seu domínio definido.

```python
# Lógica que recebe o número de partículas e as cria de forma aleatória
def criarParticulas(numParticulas: int) -> list :
    particulas: list = []
    numParticulasGeradas: int = 0

    # Gerando as particulas de forma aleatória
    while numParticulasGeradas < numParticulas :
        particulas.append(Particula())
        numParticulasGeradas += 1

    return particulas
```

Figura 4 - Criação de Partículas.

Neste trecho de código, pré-definido o número de interações pelo usuário e gerado a lista de partículas, se é utilizado a função de Schaffer em cada partícula para atualizar sua aptidão e seu pbest em cada interação, mapeando o melhor pbest de todos. Além disso, a mesma função retorna uma lista de gbests a partir de uma lista de pbests que foi gerada ao longo das interações.

```python
# Retorna uma lista de gBests a partir das particulas e a quantidade de iterações passadas como argumento
def getGbestsPorIteracao(particulas: list, numIteracoes: int) -> list :
    contadorIteracoes: int = 0
    lstGbests: list = []

    while (contadorIteracoes < numIteracoes) :
        lstPbests: list = []

        # Iterando sobre cada partícula da população atualizando sua aptidão pelo cálcula de schaffer e o seu pBest
        for i in range(len(particulas)) :
            particula: Particula = particulas[i]

            # Faz as atualizações necessárias de cada partícula em cada iteração
            particula.atualizarAptidao()
            particula.atualizarPbest()

            # Guarda as partículas pBests para depois pegar o melhor pbest de todos que no caso é o gBest
            lstPbests.append(particula.pbest)

        # Pega o melhor pBest de todos os pBests e faz uma shallow copy da sua instância atual, para ficar com o seu histórico de informações
        gBestCorrente: dict = getGBest(lstPbests)

        # Adiciona na lista de gBests a cópia do obj do melhor pBest referente a iteração corrente
        lstGbests.append(gBestCorrente)

        # Cálculo de ponderação da velocidade por iteração
        w: float = __calculoReducaoLinearPonderacaoInercia(contadorIteracoes+1, numIteracoes)

        # Iterando novamente sobre os indivíduos da população, porém atualizando velocidade e posição de acordo com o gBest encontrado
        for i in range(len(particulas)) :
            particula: Particula = particulas[i]
            particula.atualizarVelocidade(gBestCorrente, w)
            particula.atualizarPosicao()

        contadorIteracoes += 1

    return lstGbests
```

Figura 5 - Melhor gBest por interação.

3.1.2 Manipulador de arquivos (Files Handle)

A função a seguir, dado um número de interações, é criado um diretório, caso o mesmo não exista e no diretório é criado os arquivos referentes às interações, por fim, é vasculhado a melhor de todas e alterado o nome do arquivo para mostrar que é o melhor e adicionado em seu fim o valor da média dos gbests e o desvio padrão no arquivo.

```
# Variável global que armazena o local/diretório onde é guardado os arquivos
STORE_FILES_PATH: str = os.path.dirname(os.path.abspath(__file__)) + "/files"

def writeGBestData(lstGbests: list, filename: str, numero_iteracoes: int) -> None :
    try :
        directoryStorage: str = "{0}/{1}_iteracoes/".format(STORE_FILES_PATH, numero_iteracoes)

        # Cria o diretório onde armazena a partir da quantidade de iterações
        if not os.path.exists(directoryStorage) :
            os.makedirs(directoryStorage)

        with open(directoryStorage + filename, "w") as outfile :
            separador: str = ';'
            iteracao: int = 1

            # Escrevendo informações referente aos resultados de cada iteração
            cabecalho: tuple = ("Iteração", "Posição X1", "Posição X2", "Aptidão(fp)")
            outfile.write(separador.join(cabecalho) + '\n')
            for dicData in lstGbests :
                outfile.write("%d;%.16f;%.16f;%.16f\n" %(iteracao, dicData['posicao'][0], dicData['posicao'][1], dicData['aptidao']))
                iteracao += 1

            # Escrevendo o melhor gBest de todos os gBests escritos no arquivo
            bestGbestOfAll: dict = particulasPSO.getGBest(lstGbests)
            outfile.write("\nMelhor gBest:\n")
            outfile.write("Iteração: %d; Posição X1: %.16f; Posição X2: %.16f; Aptidão(fp): %.16f\n" %(lstGbests.index(bestGbestOfAll) + 1, bestGbestOfAll[

            # Escrevendo por final a média e desvio padrão da aptidão
            mediaGBests: float = particulasPSO.calculaMediaGbest(lstGbests)
            outfile.write("\nMédia(fp): %.16f\n" %(mediaGBests))
            outfile.write("Desvio Padrão(fp): %.16f\n" %(particulasPSO.calculaDesvioPadraoGbest(lstGbests, mediaGBests)))
    except IOError :
        raise Exception("Não foi possível abrir o arquivo.")
```

Figura 6 - Escreve no arquivo os gBests por iteração.

A função a seguir, dado o nome do arquivo que você quer alterar, o novo nome e o número da interação utilizado para identificar o arquivo no diretório, é utilizada para renomear um arquivo de um determinado diretório.

```
# Renomeia o arquivo de resultados passando a partir do número de iterações para pegar a sua pasta
def renameBestGBestFile(oldFileName: str, newFileName: str, numero_iteracoes: int) -> None :
    # Reescrevendo o arquivo no diretório onde se encontra todos os arquivos
    oldFileName = "{0}/{1}_iteracoes/{2}".format(STORE_FILES_PATH, numero_iteracoes, oldFileName)
    newFileName = "{0}/{1}_iteracoes/{2}".format(STORE_FILES_PATH, numero_iteracoes, newFileName)

    try :
        os.rename(oldFileName, newFileName)
    except IOError :
        raise Exception("Não foi possível renomear o arquivo. Provavelmente ele não existe.")
```

Figura 7 - Renomeação do melhor arquivo gBest.

Na próxima função, dado o número de iterações, ocorre a exclusão de todos arquivos em um diretório.

```python
# Deleta todos os arquivo de uma determina pasta baseado no número de iterações
def deleteAllFilesGBest(numero_iteracoes: int) -> None :
    try :
        directoryDeleteFiles: str = "{0}/{1}_iteracoes/".format(STORE_FILES_PATH, numero_iteracoes)
        if (os.path.exists(directoryDeleteFiles)) :
            shutil.rmtree(directoryDeleteFiles)
    except IOError :
        raise Exception("Não foi possível apagar os arquivos. Provavelmente não existe a pasta.")
```

Figura 8 - Deleta todos os arquivos gBests de um número de iterações específica.

3.1.3 User Input

Na função a seguir é determinado o número de indivíduos da população fornecido pelo usuário, assim como o número de iterações, que determina a quantidade de vezes que o algoritmo executará para o dado número de partículas, e o número de execuções que acaba sendo o número de vezes que será refeito as iterações das partículas, ou seja, acaba sendo o critério de parada do algoritmo.

```
"""
    Responsável por lidar com a entrada de dados do usuário especificamente para as partículas PSO
"""

# Determina o número de indivíduos da população que é fornecida pela entrada do usuário
def qntParticulas() -> int :
    while (True) :
        numParticulasInput: str = input("Insira o número de partículas da população a ser estudada: ")

        if (__tryParseIntPositive(numParticulasInput)) :
            return int(numParticulasInput)

# Pega o número de iterações que é o critério de parada do algoritmo e é fornecida pelo usuário
def qntIteracoesPorParticula() -> int :
    while (True) :
        numInteracoesInput: str = input("Insira o número de iterações de cada partícula na população: ")

        if (__tryParseIntPositive(numInteracoesInput)) :
            return int(numInteracoesInput)

# Determina o número de execuções do algoritmo
def qntExecucoesAlgoritmo() -> int :
    while (True) :
        numExecucoes: str = input("Insira o número de execuções do algoritmo: ")

        if (__tryParseIntPositive(numExecucoes)) :
            return int(numExecucoes)

# Checa se a string passa é possível de converter para um inteiro positivo
def __tryParseIntPositive(strInt: str) -> bool :
    if (not strInt.isdigit() or int(strInt) < 0) :
        print("Favor digite um valor inteiro válido e positivo!")
        return False

    return True
```

Figura 9 - Entrada de dados fornecidas pelo usuário.

4. Resultados

Foi realizado os testes no algoritmo para: **20 partículas**, onde elas são iteradas **50 vezes** para **10 execuções**, onde os resultados podem ser visualizados na tabela e imagem do gráfico a seguir.

Tabela 1 - Valores da média de aptidão(fp) por iteração.

Número de iterações	Valor de Aptidão(fp)
1	0,01052685725
2	0,006833611567
3	0,006276930306
4	0,01024269421
5	0,01163881984
• • •	• • •
45	0,00583290948
46	0,005852963207
47	0,005852963173
48	0,005852963169
49	0,005852963152
50	0,005832902293

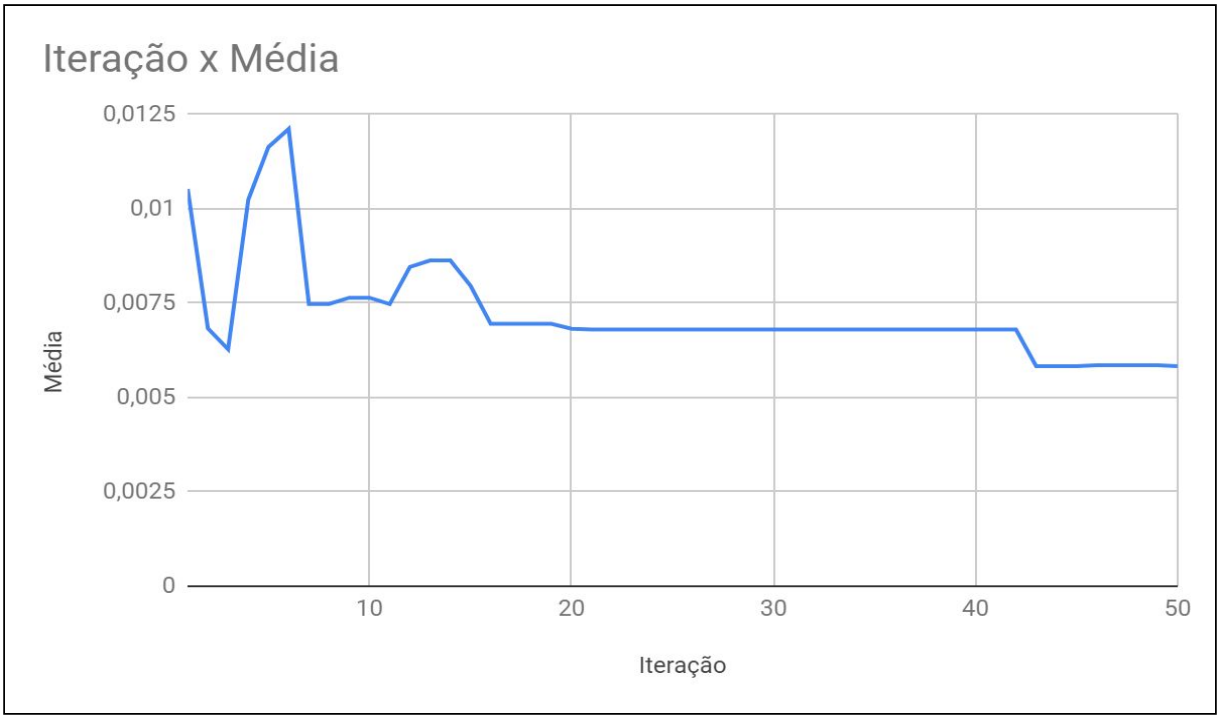


Figura 10 - Gráfico resultante da média de valores de aptidão(fp) por iteração.

Note que quanto maior o valor de iteração menor o valor de aptidão(fp), ou seja, isso mostra que mais próximo as partículas estarão próximas entre si, assim como a partícula gBest terá um melhor resultado se aproximando cada vez mais da posição (0,0).

Note também que quanto mais próximo do valor de aptidão ideal (fp = 0), as partículas vão ficando mais estáveis, como é no caso da figura 10, onde entre 20 e 42 iterações o valor da média de cada iteração por execuções do algoritmo, onde as partículas variam minimamente.

Outro ponto importante é a variação que se tem nas primeiras 17 iterações até a estabilização das partículas. Isso se dá porque as partículas estão sempre atualizando seu pBest, assim como o algoritmo o seu gBest, fazendo com que tenha uma maior variação até seu ponto de estabilização. O que poderia também melhorar essa grande variação inicial é a quantidade de partículas, dado os resultados são encontrados se baseado no gBest.

5. Referências

- https://pt.wikipedia.org/wiki/Optimiza%C3%A7%C3%A3o_por_enxame_de_part%C3%ADculas