presentation

December 16, 2020

1 *Aplicação de PSO Híbrido* :

2 *Caixeiro Viajante*

2.1 *Overview*

2.1.1 PSO Hibrido + Fator Genético

- **Posição da Partícula:** Uma rota válida
- **Velocidade:** N par de trocas simples entre elementos da rota
- **Fator Genético**: Cada partícula tem 50% de chance de desencadear evento genético que substitui os 2 piores elementos do enxame por 2 partículas da próxima geração
  - **Seleção dos pais:** GENITOR
  - **Cross-over**: Order Cross-over (OX1)
  - **Sem mutação**

2.1.2 Partícula

[1]:
```python
import numpy as np
class DParticle:
    def __init__(self, path: np.array):
        self.position = path
        self.combination_count = np.random.randint(len(path) * (len(path) - 1))␣
 ↪+ 1
        self.velocity = np.random.randint(len(path), size=(self.
 ↪combination_count, 2))
        self.best_position = np.copy(self.position)
        self.best_path_len = np.inf

    def __repr__(self):
        return self.__str__()
```

2.1.3 Fit

[2]:
```python
def fit(path, problem):
    cyclic_path = np.hstack((path, np.array([path[0]])))
    return sum(problem.get_weight(a, b) for a, b in zip(cyclic_path[0:],␣
 ↪cyclic_path[1:]))
```

2.1.4 Atualização

[3]:
```
def discrete_velocity(particle: DParticle):
    return random.choices(particle.velocity, k=np.random.randint(len(particle.
 ↪position)))
```

2.1.5 Algoritmo PSO Discreto Híbrido

[4]:
```
def submit(self, iterations=1000):
    for i in range(iterations):
        for particle in self.particles:
            distance = fit(particle.position, self.problem)
            logger.debug(f"Distance: {distance}\t Path:{particle.position}\tV:
 ↪{particle.velocity}")

            # Is it the best particle distance so far?
            if distance < particle.best_path_len:
                particle.best_position = np.copy(particle.position)
                particle.best_path_len = distance
                # May be the best global distance as well?
                if distance < self.best_path:
                    self.best_path = distance
                    self.best_path_pos = np.copy(particle.position)
                    logger.info(f"Best distance: {self.best_path}\tBest Path:
 ↪{self.best_path_pos}")
            # Adjust position
            velocity = discrete_velocity(particle)
            adjust_discrete_position(particle, velocity)

            # Adding genetic vector
            if random.random() <= 0.5:
                parents = self.parent_extractor.extract_parent(problem=self.
 ↪problem, population=self.particles)
                offspring = self.crossover.cross(parents, 2, self.problem)
                self.particles.extend(DParticle(off) for off in offspring)
                natural_select(problem=self.problem, population=self.particles,␣
 ↪die=len(offspring))
```

2.1.6 Auxiliares

[5]:
```
# Realiza as trocas entre posições, similar a mutação SIM
def adjust_discrete_position(particle, velocity):
    for exchange in velocity:
        tmp = np.copy(particle.position[exchange[0]])
        particle.position[exchange[0]] = particle.position[exchange[1]]
        particle.position[exchange[1]] = tmp
```

2.2 Resultados

[6]:
```
import pandas as pd
pd.read_csv("benchmark.csv")
```

[6]:

	algoritmo	problem	mean	min
0	Ideal	24	-	1272
1	AG Sugerido	24	1331	1272
2	AG Desenvolvido	24	-	1300
3	DPSO Hibrido	24	1471.5	1307
4	PSO Discreto	24	2125.6	1813
5	PSO Continuo	24	2148.3	2021
6	-	-	-	-
7	Ideal	48	-	5046
8	AG Sugerido	48	5533	5080
9	AG Desenvolvido	48	-	6893
10	DPSO Hibrido	48	7924.3	7293
11	PSO Continuo	48	15801	14630
12	PSO Discreto	48	16074	15422

2.3 Referências

[1] PARTICLE SWARM OPTIMIZATION FOR SOLVING CONSTRAINT SATISFACTION PROBLEMS,Lin., https://core.ac.uk/download/pdf/56374467.pdf