UNIVERSIDADE FEDERAL DE SÃO PAULO
CAMPUS SÃO JOSÉ DOS CAMPOS
INSTITUTO DE CIÊNCIA E TECNOLOGIA

Computação Bioinspirada

Problema do Caixeiro Viajante

Alunos: Álvaro Cardoso Vicente de Souza RA: 133536

Professor: Dr. Márcio Basgalupp

São José dos Campos - SP
Novembro 2022

Resumo

O relatório a seguir aborda três implementações diferentes para o problema do caixeiro viajante (traveling salesman problem) onde buscamos encontrar o caminho ótimo a ser percorrido entre oito cidades, de maneira a minimizar a distância a ser percorrida visitando todas as cidades e retornando a cidade inicial. A primeira implementação utiliza uma abordagem de algoritmos genéticos, a segunda implementação utiliza uma abordagem de inteligência coletiva, mais especificamente, inteligência de enxame. A terceira abordagem, por sua vez, utiliza um algoritmo baseado em colônias de formigas. Durante o decorrer do texto serão abordadas as representações utilizadas nas implementações assim como os resultados obtidos para cada um dos algoritmos.

1. Algoritmo Genético - GA

1.1 Representação

Para esse algoritmo o problema foi representado utilizando uma matriz de adjacência que contém as distâncias para cada uma das oito cidades, como demonstrada na Figura 1, abaixo:

```
city_cost = np.array
([
        [0, 42, 61, 30, 17, 82, 31, 11],
        [42, 0, 14, 87, 28, 70, 19, 33],
        [61, 14, 0, 20, 81, 21, 8, 29],
        [30, 87, 20, 0, 34, 33, 91, 10],
        [17, 28, 81, 34, 0, 41, 34, 82],
        [82, 70, 21, 33, 41,0, 19, 32],
        [41, 19, 8, 91, 34, 19, 0, 59],
        [11, 33, 29, 10, 82, 32, 59, 0],
    ])
```

Figura 1 - Matriz de adjacência das cidades utilizadas no problema.

Cada indivíduo da população inicial consistia basicamente em um vetor de nove elementos, gerados a partir de uma permutação aleatória das 8 cidades, onde a última posição era a cidade inicial. A cada geração, toda a população se organizava em pares para que os novos indivíduos fossem gerados a partir das informações dos pais. Os indivíduos eram gerados seguindo a regra ilustrada na Figura 2, onde o meio do vetor era de um pai e o início e o final do vetor era de outro pai seguindo a ordem em que as cidades aparecem e evitando repetição. Existia uma pequena chance dos indivíduos sofrerem mutações onde duas cidades eram trocadas de posição e os indivíduos podiam também ser uma cópia de seus pais.

```
pai1 =   [5 6 2 1 7 0 4 3 5]
pai2 =   [6 1 2 3 4 0 7 5 6]
filho1=  [6 3 2 1 7 0 4 5 6]
filho2=  [5 6 2 3 4 0 1 7 5]
```

Figura 2 - Pais e Filhos.

Após cada cruzamento, cada indivíduo da nova população tinha seu valor do *fitness* verificado e o menor valor até o momento era salvo junto a rota que gerou aquele valor. O *fitness* era calculado somando a distância de cada cidade seguindo a ordem de cada indivíduo, como ilustrado na figura 3.

```
def fitness(gene):
    return sum(
    [
        city_cost[gene[i], gene[i + 1]]
        for i in range(len(gene) - 1)
    ])
```

Figura 3 - Cálculo do *Fitness*

1.2 Execução e Soluções

Para esse algoritmo foram feitas três simulações distintas com 30 execuções cada, a primeira levou em conta uma população de 500 indivíduos, se estendendo por 10000 gerações, com uma chance de mutação de 15% e 20% de chance dos filhos serem iguais aos pais. Com esses parâmetros, obtemos o seguinte resultado:

Melhor Indivíduo	Fitness	Média Fitness	Desvio Padrão Fit
[7, 3, 5, 6, 2, 1, 4, 0, 7]	140	155.6	13.04

Tabela 1 - Resultados Obtidos GA 1

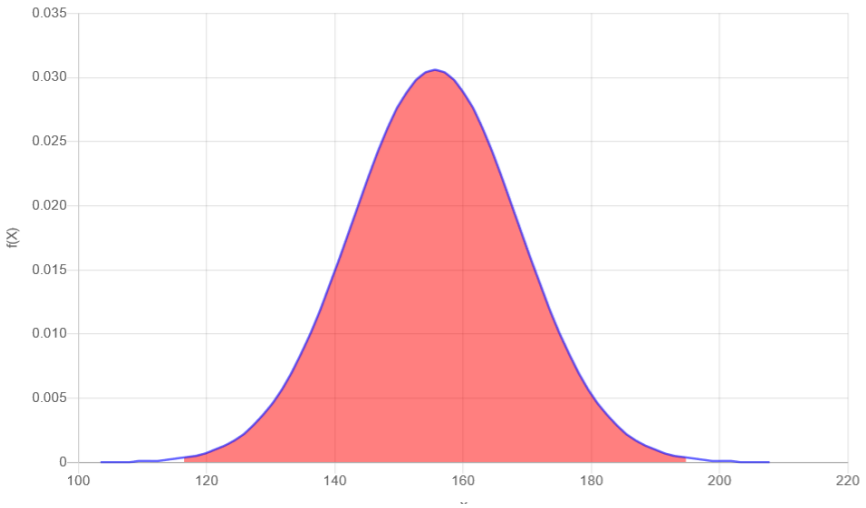

Figura 4 - Distribuição Normal GA 1

A segunda execução levou em conta uma população de 50 indivíduos, se estendendo por 100 gerações, com uma chance de mutação de 5% e 10% de chance dos filhos serem iguais aos pais. Com esses parâmetros, obtemos o seguinte resultado:

Melhor Indivíduo	Fitness	Média Fitness	Desvio Padrão Fit
[7, 0, 4, 1, 2, 6, 5, 3, 7]	140	185.6	19.11

Tabela 2 - Resultados Obtidos GA 2

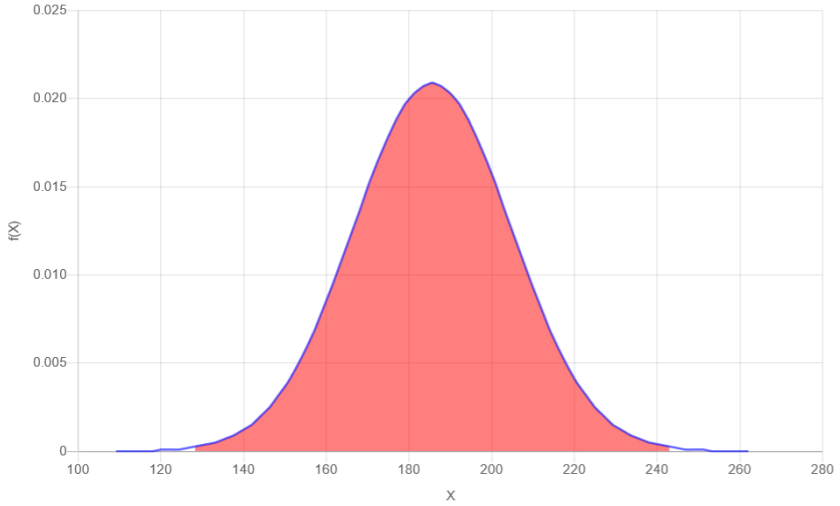

Figura 5 - Distribuição Normal GA 2

A terceira execução levou em conta uma população de 1000 indivíduos, se estendendo por 10000000 gerações, com uma chance de mutação de 10% e 10% de chance dos filhos serem iguais aos pais. Com esses parâmetros, obtemos o seguinte resultado:

Melhor Indivíduo	Fitness	Média Fitness	Desvio Padrão Fit
[3, 5, 6, 2, 1, 4, 0, 7, 3]	140	146.7	9.87

Tabela 3 - Resultados Obtidos GA 3

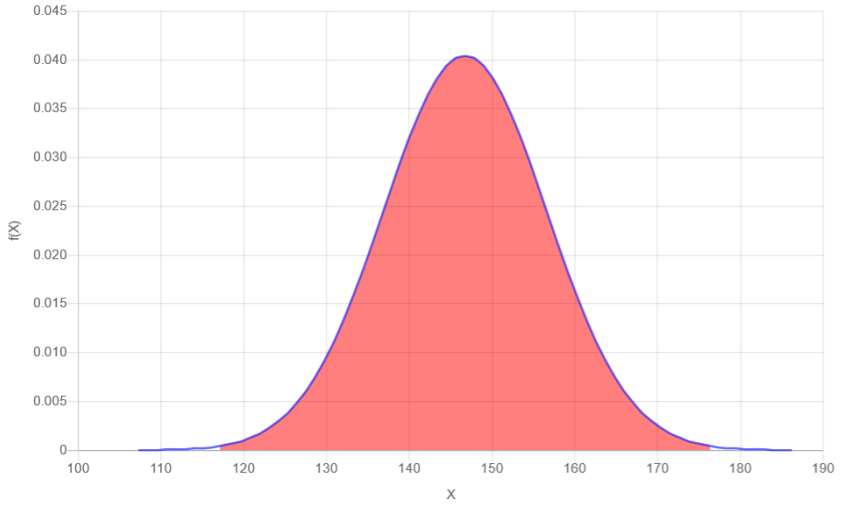

Figura 6 - Distribuição Normal GA 3

1.3 Discussão dos resultados

Percebe-se com as simulações que o tamanho da população e o número de gerações tem os maiores impactos nos resultados obtidos onde a simulação com maior população e maior número de gerações obteve um resultado com um menor desvio padrão, ou seja, tem uma maior probabilidade de chegar a solução ótima de 140 de custo , ainda assim, não podemos esquecer de mencionar a influência da mutação que permite que novos indivíduos que divergem das características de seus

pais fazendo com que possamos cobrir um maior espaço de busca nas possíveis soluções diminuindo a chance de convergir para um ótimo local.

Com relação ao resultado, acredita-se que o fato de soluções não ótimas ainda aparecem mesmo quando aumentamos o número de gerações e tamanho da população, se deve muito a característica do algoritmo e da maneira que foi escolhida para a geração dos indivíduos com base nas informações dos pais, que não garante que seus filhos trazem consigo soluções melhores para o problema, ainda assim, acredita-se que ainda é possível obter resultados melhores alterando os parâmetros da simulação.

2. Inteligência de Enxame - PSO

2.1 Representação

Para esse algoritmo foi utilizado como base o artigo Particle swarm optimization for traveling salesman problem (KANGPING WANG, LAN HUANG, CHUN-GUANG ZHOU, WE1 PhG) e o problema foi representado utilizando uma matriz de adjacência que contém as distâncias para cada uma das oito cidades, como demonstrada na Figura 4, abaixo:

```
adjacency_mat = np.array(
    [
        [0, 42, 61, 30, 17, 82, 31, 11],
        [42, 0, 14, 87, 28, 70, 19, 33],
        [61, 14, 0, 20, 81, 21, 8, 29],
        [30, 87, 20, 0, 34, 33, 91, 10],
        [17, 28, 81, 34, 0, 41, 34, 82],
        [82, 70, 21, 33, 41,0, 19, 32],
        [41, 19, 8, 91, 34, 19, 0, 59],
        [11, 33, 29, 10, 82, 32, 59, 0],
    ])
```

Figura 7 - Matriz de adjacência das cidades utilizadas no problema.

Cada indivíduo da população inicial consistia basicamente em uma partícula que tinha a rota de sua coordenada, a *swap sequence* que pode ser abstraída como sua velocidade de deslocamento, *fitness* da rota de sua posição, melhor rota e melhor *fitness* encontrados até o momento. Onde a rota das posições das partículas pertencentes a população inicial era uma permutação aleatória das oito cidades. A cada iteração os indivíduos se deslocavam para a melhor rota global ou local dependendo de um valor aleatório ser menor do que as constantes *phi1* ou *phi2*, para evitar convergir para ótimos locais e não globais. Os valores de *phi1* e *phi2* também eram usados para decidir se determinado ponto iria fazer parte da nova rota de deslocamento da partícula ou seja, a *swap_sequence* gerada poderia sofrer alterações antes de ser passada aos indivíduos.

A *basic swap sequence*, ilustrada na figura 5, consiste basicamente em instruções das posições que devem ser trocadas para que uma sequência se torne outra, por exemplo, para a sequência 1, 2, 3, 4 temos que a SS seria S_{01}, S_{23} caso a sequência que queremos gerar fosse 2, 1, 4, 3. Assim, podemos simular um “deslocamento a uma certa velocidade” de uma rota (posição) a outra.

```
def basic_ss(r1, r2):
    cp_r2 = r2.copy()
    result_ss = []
    for i in range (len(r1)):
        if r1[i] == r2[i]:
            return result_ss
        index = cp_r2.index(r1[i])
        result_ss.append((i, index))
        cp_r2[i], cp_r2[index] = cp_r2[index], cp_r2[i]
    return result_ss
```

Figura 8 - Função que gera a basic swap sequence.

O *fitness*, assim como no exemplo anterior, era calculado somando a distância de cada cidade seguindo a ordem da rota de cada partícula e no final o melhor fitness dentre todas as partículas era mostrado.

2.2 Execução e Soluções

Para esse algoritmo foram feitas três simulações distintas com 30 execuções cada, a primeira levou em conta uma população de 10 partículas, por 100 iterações, com *phi1* = 0.66 e *phi2* = 0.33. Com esses parâmetros, obtemos o seguinte resultado:

Melhor Rota	Fitness	Média Fitness	Desvio Padrão Fit
[5, 6, 2, 1, 4, 0, 7, 3, 5]	140	151.23	11.29

Tabela 4 - Resultados Obtidos PSO 1

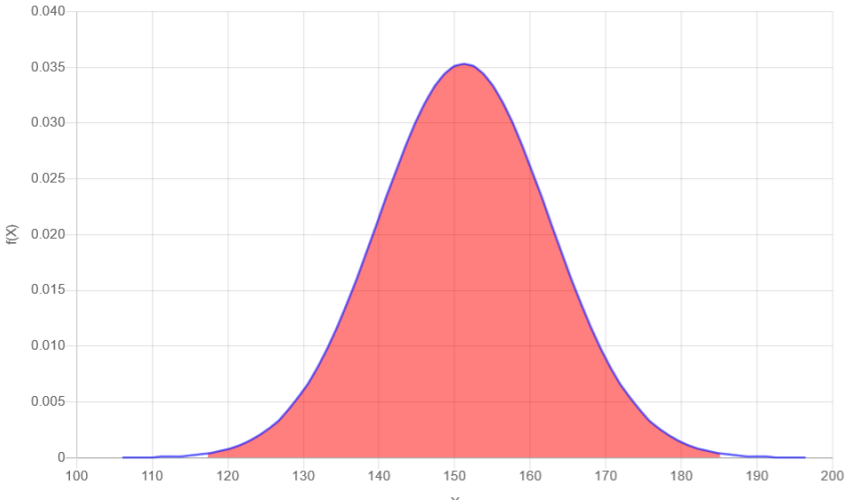

Figura 9 - Distribuição Normal PSO 1

A segunda execução levou em conta uma população de 50 partículas , por 1000 iterações, com *phi1* = 0.66 e *phi2* = 0.33. Com esses parâmetros, obtemos o seguinte resultado:

Melhor Rota	Fitness	Média Fitness	Desvio Padrão Fit
[3, 7, 0, 4, 1, 2, 6, 5, 3]	140	140	0

Tabela 5 - Resultados Obtidos PSO 2

A terceira execução levou em conta uma população de 50 partículas , por 1000 iterações, com *phi1* = 0.22 e *phi2* = 0.78. Com esses parâmetros, obtemos o seguinte resultado:

Melhor Rota	Fitness	Média Fitness	Desvio Padrão Fit
[6, 2, 1, 4, 0, 7, 3, 5, 6]	140	166.93	15.82

Tabela 6 - Resultados Obtidos PSO 3

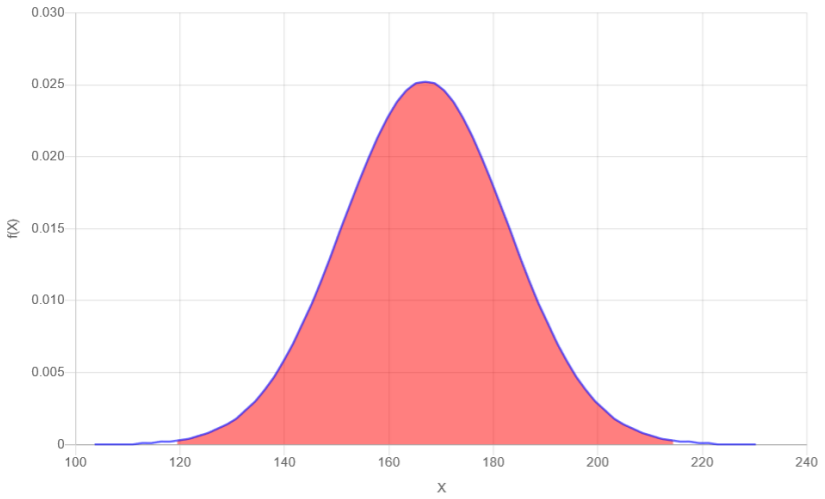

Figura 10 - Distribuição Normal PSO 3

2.3 Discussão dos resultados

Assim como no exemplo anterior da implementação do algoritmo genético, podemos perceber que o número de partículas e o número de iterações influenciam bastante para encontrarmos uma solução ótima de custo 140. Outro fator que podemos destacar também é que os parâmetros *phi1* e *phi2* tem um grande influencia para onde a solução vai convergir já que eles influenciam diretamente na *swap sequence* e na probabilidade de focarmos em ótimos globais e locais, onde a melhor solução parece ser uma mais voltada ao ótimos globais, sem negligenciar os locais com um número maior de partículas e interações, como ilustrado no segundo exemplo onde o desvio padrão é 0.

3. Colônia de Formigas - ACO

3.1 Representação

Para esse algoritmo, assim como nas versões anteriores, o problema foi representado utilizando uma matriz de adjacência que contém as distâncias para cada uma das oito cidades, ou seja, um grafo com arestas que possuem pesos correspondentes às distâncias de cada cidade. Uma matriz heurística (1/mat_adj) também foi utilizada para o cálculo da matriz de probabilidade de cado *node*.

Temos como parâmetros o número de formigas, taxa de evaporação do feromônio, taxa de intensificação do feromônio, peso do aumento do feromônio a cada iteração representado pelo alfa e a probabilidade de escolher o melhor caminho.

Ao iniciarmos o algoritmos utilizamos a função de fit, que recebe a matriz de adjacência, o número de iterações, número de repetições para parada antecipada e se queremos maximizar ou minimizar o problema. A lista dos *nodes* disponíveis é criada e as formigas são colocadas em um *node* aleatório e a cada iteração escolhem o próximo *node* levando em conta a probabilidade de se escolher o melhor caminho, se não escolher o melhor caminho é feita uma escolha entre os outros caminhos levando em conta os feromônios de cada caminho.

Cada formiga percorre o grafo escolhendo entre os *nodes* disponíveis e ao chegarmos ao final da iteração, os feromônios são evaporados, o melhor caminho tem seus feromônios atualizados e as probabilidades são atualizadas seguindo a equação (self.pheromone_matrix ^ self.heuristic_alpha) * (self.heuristic_matrix). O melhor fitness, que é calculado somando a distância de cada rota, e a melhor rota rota são salvos para serem atualizados na próxima iteração. Nas próximas iterações as formigas novamente começam em um *node* aleatório e vão escolhendo os próximos *nodes* de acordo com os valores atualizados de feromônios e da matriz de probabilidade.

Ao chegarmos ao número determinado de repetições no valor do fitness ou ao final das iterações o algoritmo é finalizado.

3.2 Execução e Soluções

Para esse algoritmo foram feitas três simulações distintas com 30 execuções cada, a primeira levou em conta uma população de 8 formigas, por 100 iterações, com intensificação do feromônio de 1, alpha 1, probabilidade de escolher melhor = 10% e taxa de evaporação de 30%.

Melhor Rota	Fitness	Média Fitness	Desvio Padrão Fit
[2, 1, 4, 0, 7, 3, 5, 6, 2]	140	142.03	5.59

Tabela 7 - Resultados Obtidos ACO 1

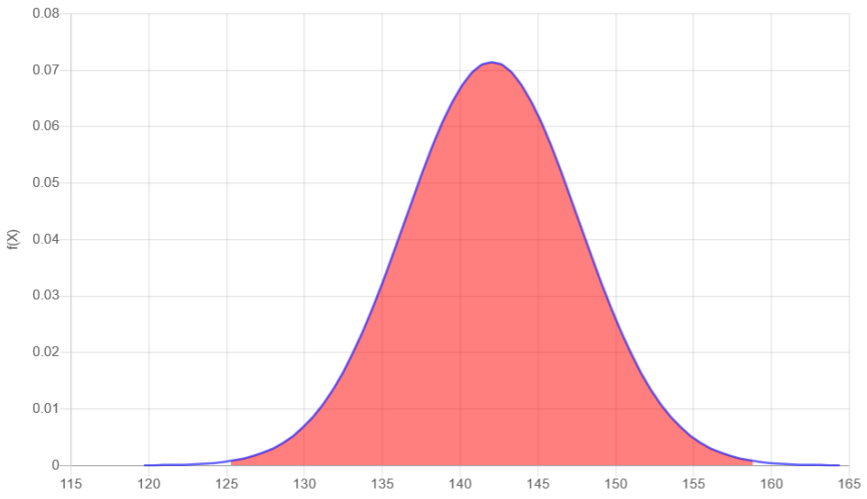

Figura 11 - Distribuição Normal ACO 1

A segunda execução levou em conta uma população de 24 formigas, por 100 iterações, com intensificação do feromônio de 1, alpha 1, probabilidade de escolher melhor = 10% e taxa de evaporação de 30%.

Melhor Rota	Fitness	Média Fitness	Desvio Padrão Fit
[4, 1, 2, 6, 5, 3, 7, 0, 4]	140	140	0

Tabela 8 - Resultados Obtidos ACO 2

A terceira execução levou em conta uma população de 8 formigas, por 100 iterações, com intensificação do feromônio de 2, alpha 1, probabilidade de escolher melhor = 10% e taxa de evaporação de 70%.

Melhor Rota	Fitness	Média Fitness	Desvio Padrão Fit
[2, 6, 5, 3, 7, 0, 4, 1, 2]	140	154.5	11.90

Tabela 9 - Resultados Obtidos ACO 3

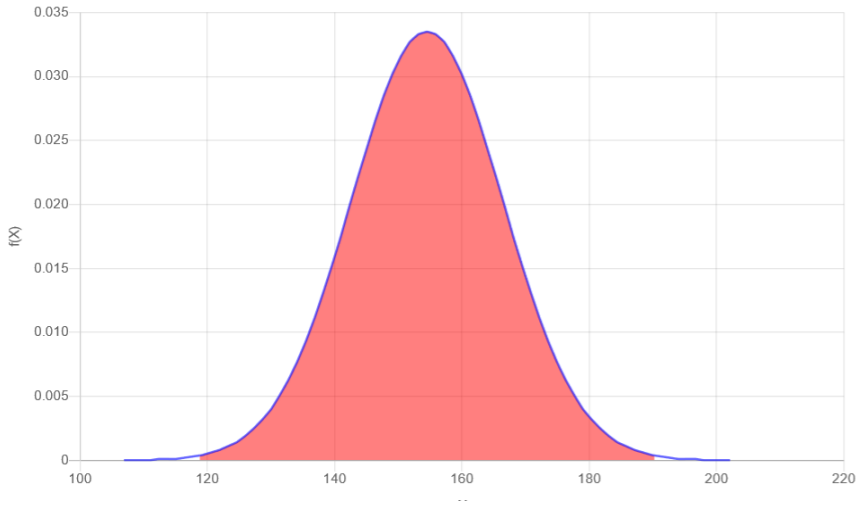

Figura 12 - Distribuição Normal ACO 3

3.3 Discussão dos resultados

Para essa fica claro que a taxa de evaporação do feromônio, o número de formigas e a intensificação contribuem bastante para chegarmos a uma ótima solução. Assim como no exemplo do PSO, chegamos a uma execução onde observamos um desvio padrão de 0, justamente onde o número de formigas era maior. Para problemas mais complexos acredito que o número de iterações também tenha um impacto maior, assim como, a probabilidade de escolha do melhor caminho que pode fazer com que a solução tenda a convergir para um ótimo local.

Outro ponto interessante a ser destacado é o fato de uma taxa de evaporação do feromônio maior ter um impacto muito grande na qualidade das soluções encontradas, apresentando um desvio padrão de aproximadamente 12.

4. Conclusão

A respeito desse trabalho podemos concluir que existem diversas maneiras de se representar o mesmo problema e chegar a soluções satisfatórias e que os parâmetros escolhidos tem um grande influência no resultado final obtido.

Ainda podemos destacar que apesar de todas as implementações chegarem a soluções ótimas em uma das 30 execuções em cada um dos três casos, o algoritmo PSO e ACO tiveram um desempenho superior quando comparados ao algoritmos genético, já que, ambos os algoritmos tiveram um teste onde os resultados obtidos nas trinta execuções foram a solução ótima.

5. Referências Bibliográficas

Particle swarm optimization for traveling salesman problem - KANGPING WANG, LAN HUANG, CHUN-GUANG ZHOU, WEI PANG. Disponível em: <https://ieeexplore.ieee.org/document/1259748>

Ant-Colony-Optimization - johnberroa.

Disponível em: <https://github.com/johnberroa/Ant-Colony-Optimization>