Universidade Federal do Pampa
Campus Alegrete

Engenharia de Software
Projeto e Análise de Algoritmos, 2021/2
Prof. Dr. Rodrigo Mansilha

Avaliação Analítica

nome: Daniel Oliveira Freitas

matrícula: 2010101289

email: danielodf2.aluno@unipampa.edu.br

link repositório:
https://github.com/Daniel-Oliveira-de-Freitas/TSP_ACO_Python.git

Problema Escolhido

Problemas de otimização combinatória são difíceis de resolver, isto é, o seu custo computacional cresce exponencialmente com o aumento da entrada. É o caso, por exemplo, do problema do caixeiro viajante (Traveling Salesman Problem - TSP). Imagine um conjunto de cidades completamente conectadas entre si, isto é, para cada par de cidades, há uma estrada que as liga. O problema consiste, então, em encontrar o menor caminho para percorrer todas as cidades uma única vez.

Justificativa do Problema

O conjunto de todos os caminhos possíveis definem o espaço de busca para este problema. Para conjuntos de poucas cidades, até cinco ou seis, podemos testar todas as possibilidades para encontrar o menor caminho. Mas o problema se torna computacionalmente intratável para conjuntos maiores de cidades. Tal fato despertou a necessidade de se elaborar estratégias computacionalmente baratas, mas que encontrassem soluções ótimas ou próximas delas para esse tipo de problema. Essa é a ideia principal por trás das metaheurísticas. São estratégias para explorar o espaço de busca de maneira eficiente, encontrando soluções ótimas, próximas da melhor solução, mas sem ter de explorar o espaço de busca exaustivamente. Uma metaheurística, enquanto estratégia mais geral para obter soluções ótimas a um custo razoável, pode se basear em uma das duas abordagens ou em ambas conjuntamente. ACO, como veremos, é primordialmente uma metaheurística baseada na construção de solução, embora ela possa ser combinada também com buscas locais.

Algoritmo que Servirá de Base para Resolver o Problema

A otimização por colônia de formigas (Ant Colony Optimization - ACO) é uma metaheurística para a solução de problemas combinatórios. Ela é inspirada no comportamento de formigas na busca de alimentos. Quando uma formiga precisa decidir para onde ir, ela usa informação proveniente de feromônio previamente depositado por outras formigas que passaram por aquele local. A direção que tiver maior depósito de feromônio será escolhida pela formiga. Por este processo de busca, formigas são capazes de encontrar o menor caminho de uma fonte de comida para o seu ninho.

Justificativa do Algoritmo

Para o problema do caixeiro viajante, ela funciona assim. Dado um conjunto de N cidades, escolhe-se uma delas e a coloca na solução. Avalia-se, então, qual das cidades restantes está mais próxima daquela e ela é adicionada também na solução. Esse processo é repetido N vezes, até que se tenha um caminho completo, isto é, uma solução para o problema do caixeiro viajante. Repare, no entanto, que se tivermos N cidades, teremos apenas N soluções gulosas distintas. Uma vez escolhida a cidade inicial, a construção da solução gulosa torna-se completamente determinística. O problema, como já dissemos, consiste em encontrar o menor caminho para percorrer um conjunto de cidades que estão completamente conectadas entre si. Neste trabalho, trabalhamos apenas com a versão simétrica do problema. Cada cidade é associada a um nó do grafo de construção das formigas. A distância entre uma cidade **i** e **j** é chamada de dij . Assim, uma instância do TSP é um grafo (N, E), onde N é o conjunto de cidades e E o conjunto de arestas entre as cidades.

Conjunto de Parâmetros para Avaliação Experimental

Será utilizado como parâmetro para avaliação experimental, o número de formigas, o número de cidades e o número de viagens e a redução e aumento de feromônio .

Conjunto de Valores para Avaliação Experimental

Para cada um dos parâmetros, faremos uma avaliação experimental e utilizaremos uma quantidade x de valores e os valores restantes informaremos de forma padronizada para compararmos o impacto de diferentes entradas com valores padrões.

Valores Experimentais
número de cidades: 45 , 75 , 95 , 150 , 300
número de formigas: 35 , 65 , 85 , 110 , 200
número de viagens: 7 , 10 , 15 , 20 , 30 , 50

Valores Padrões
número de cidades: 150
número de formigas: 110
número de viagens: 30

Métricas de Desempenho

Para as métricas de distância, número de viagens e de formigas farei uma avaliação do tempo de processamento com a variação dos valores dos parâmetros com a utilização da meta-heuristica colônia de formigas em relação a otimização das rotas através do feromônio.

Implementação

A implementação dos experimentos foi realizada na linguagem Java onde foi utilizado métodos para executar os parâmetros utilizados diretamente nos atributos facilitando os testes e evitando redundância pois mudamos apenas em 3 linhas não tendo a necessidade de fazer isso repetidamente em vários locais do código.
Script para orquestrar a realização de experimentos.

parâmetros que serão testados com valores experimentais

```java
public class CaixeiroViajanteFormigas {
    private static final int numCidades = (n);
    private static final int numFormigas = (n);
    private static final int viagens = (n);
```

Cria rotas iniciais aleatórias para cada formiga Complexidade: 2.O(n^2).

```java
private static void criaDistancias() {
    int dist;
    distancias = new int[numCidades][numCidades];
    for (int i = 0; i < numCidades; i++) {
        for (int j = i; j < numCidades; j++) {
            if (i == j) {
                dist = 0;
            } else {
                dist = random.nextInt(20) + 1;
            }
            distancias[i][j] = dist;
            distancias[j][i] = dist;
        }
    }
}
```

Cria rotas iniciais aleatórias para cada formiga complexidade: 2.O(n^2).

```java
private static void distribuiFormigas() {
    formigas = new int[numFormigas][numCidades + 1];
    for (int i = 0; i < numFormigas; i++) {
        for (int j = 0; j < numCidades; j++) {
            formigas[i][j] = j;
        }
    }
    for (int i = 0; i < numFormigas; i++) {
        for (int j = 0; j < numCidades; j++) {
            int dest = random.nextInt(numCidades);
            if (dest != j) {
                int aux = formigas[i][j];
                formigas[i][j] = formigas[i][dest];
                formigas[i][dest] = aux;
            }
        }
        formigas[i][numCidades] = formigas[i][0]; // volta para o inicio
    }
}
```

Calcula a distância de uma rota complexidade: O(n)

```java
private static int distanciaPercorrida(int[] cidades) {
    int ret = 0;
    for (int i = 0; i < cidades.length - 1; i++) {
        ret += distancias[cidades[i]][cidades[i + 1]];
    }
    return ret;
}
```

Atualiza a melhor rota avaliando a distância percorrida pelas formigas complexidade: O(n)

```
private static void melhorRota() {
    for (int i = 0; i < numFormigas; i++) {
        int dist = distanciaPercorrida(formigas[i]);
        if (melhorDistancia == 0 || dist < melhorDistancia) {
            if (melhorRota == null) {
                melhorRota = new int[numCidades + 1];
            }
            melhorDistancia = dist;
            arraycopy(formigas[i], 0, melhorRota, 0, formigas[i].length);
        }
    }
}
```

Inicializa a quantidade de feromônios nos trajetos complexidade: O(n^2)

```
private static void inicializaFeromonios() {
    feromonios = new double[numCidades][numCidades];
    for (int i = 0; i < numCidades; i++) {
        for (int j = 0; j < numCidades; j++) {
            feromonios[i][j] = 0.01;
        }
    }
}
```

Inicia uma nova viagem das formigas
complexidade: O(n) + complexidade da função viagemFormiga

```
private static void atualizaRotasFormigas() {
    for (int i = 0; i < numFormigas; i++) {
        cidade atual da formiga
        int cidade = formigas[i][numCidades - 1];
        int cidade = random.nextInt(numCidades);
        formigas[i] = viagemFormiga(cidade);
    }
}
```

Cria uma rota a partir de uma cidade inicial
complexidade: O(n) + complexidade da função proximaCidade

```java
private static int[] viagemFormiga(int inicio) {
    boolean[] visitadas = new boolean[numCidades];
    int[] novaRota = new int[numCidades + 1];
    novaRota[0] = inicio;
    visitadas[inicio] = true;
    for (int i = 0; i < numCidades - 1; i++) {
        int origem = novaRota[i];
        int destino = proximaCidade(origem, visitadas);
        novaRota[i + 1] = destino;
        visitadas[destino] = true;
    }
    novaRota[numCidades] = novaRota[0];
    return novaRota;
}
```

Define a próxima cidade a ser visitada levando em consideração os feromônios e a distância do trajeto complexidade: O(n^3) + O(n/2)

```java
private static int proximaCidade(int origem, boolean[] visitadas) {
    double[] aux = new double[numCidades];
    double soma = 0.0;
    for (int i = 0; i < numCidades; i++) {
        if (i != origem && !visitadas[i]) {
            aux[i] = pow(feromonios[origem][i], alpha) * pow(0.1 / distancias[origem][i], beta);
            soma += aux[i];
        }
    }
    for (int i = 0; i < numCidades; i++) {
        aux[i] /= soma;
    }
    double[] acum = new double[numCidades + 1];
    for (int i = 0; i < numCidades; i++) {
        acum[i + 1] = acum[i] + aux[i];
    }
    double p = random.nextDouble();

    for (int i = 0; i < acum.length - 1; ++i) {
        if (p >= acum[i] && p < acum[i + 1]) {
            return i;
        }
    }
    return 0;
}
```

Atualiza a tabela de feromônios complexidade: O(n^2) * O(m) * complexidade da função distanciaPercorrida * complexidade da função formigaFezCaminho

```java
private static void atualizaFeromonios() {
    for (int i = 0; i < numCidades; i++) {
        for (int j = 0; j < numCidades; j++) {
            if (j == i) {
                continue;
            }
            for (int f = 0; f < numFormigas; f++) {
                int dist = distanciaPercorrida(formigas[f]);
                double reducao = feromonios[i][j] * (1 - reducaoFeromonio);
                double aumento = 0.0;
                if (formigaFezCaminho(f, i, j)) {
                    aumento = aumentoFeromonio / dist;
                }
                feromonios[i][j] = reducao + aumento;
                if (feromonios[i][j] < 0.0001) {
                    feromonios[i][j] = 0.0001;
                }
            }
        }
    }
}
```

Verifica se a formiga fez o trajeto da cidade origem a destino complexidade: O(n)

```java
private static boolean formigaFezCaminho(int formiga, int origem, int destino) {
    if (origem == destino) {
        return false;
    }
    for (int i = 0; i < numCidades + 1; i++) {
        if (origem == formigas[formiga][i]) {
            if (i == 0 && formigas[formiga][i + 1] == destino) {
                return true;
            } else if (i == numCidades - 1 && formigas[formiga][i - 1] == destino) {
                return true;
            } else if (i > 0 && i < numCidades - 1) {
                if (formigas[formiga][i - 1] == destino || formigas[formiga][i + 1] == destino) {
                    return true;
                }
            }
            return false;
        }
    }
    return false;
}
```

Armazena o melhores caminhos para desenho do gráfico

```java
private static void armazenaResultados(int numViagem) {
    if (resultados == null) {
        resultados = new LinkedHashMap<Integer, Integer>();
    }
    resultados.put(numViagem + 1, distanciaPercorrida(melhorRota));
}
```

imprime rota e melhor rota

```
private static void imprimeRotas() {
    for (int i = 0; i < numFormigas; i++) {
        out.printf("Formiga %2d fez a rota: %s de distancia %d\n", i, Arrays.toString(formigas[i]),
                distanciaPercorrida(formigas[i]));
    }
}

private static void imprimeMelhorRota() {
    out.printf("melhor rota: %s de distancia %d\n", Arrays.toString(melhorRota), distanciaPercorrida(melhorRota));
}
```

método que gera o gráfico apresentado no final da execução do algoritmo.

```
private static void apresentaGrafico() {
    JFrame jf = new JFrame("Grafico das Formigas");
    JLabel f = new JLabel();
    JLabel nf = new JLabel();
    long tempoFinal = System.currentTimeMillis();
    jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    XYSeries xy = new XYSeries("Distancia");
    XYSeries xtt = new XYSeries(String.valueOf(tempoFinal));
    for (Integer v : resultados.keySet()) {
        xy.add(v, resultados.get(v));
    }
    XYSeriesCollection col1 = new XYSeriesCollection(xtt);
    XYSeriesCollection col = new XYSeriesCollection(xy);
    JFreeChart jfc = ChartFactory.createXYLineChart("ACO", "viagem", "Distância", col, PlotOrientation.VERTICAL,
            true, true, false);
    ChartPanel cp = new ChartPanel(jfc);
    nf.setText("Paramentros: "
            + " | Núm.de Cidades:" + String.valueOf( numCidades )
            + " | Núm.de Formigas: " + String.valueOf( numFormigas )
            + " | Núm.de Viagens: " + String.valueOf( viagens ));
    f.setText(" | Tempo de execução: " + String.valueOf(tempoFinal/1000000000) + " seg");
    jf.add(cp);
    jf.add(nf);
    jf.add(f);
    jf.pack();
    jf.setLayout(new GridLayout());
    jf.setLocationRelativeTo(null);
    jf.setVisible(true);
}
```

experimentos

Como base para os experimentos foram usados os parâmetros: número de formigas: 11, número de cidades: 15, número de viagens: 3, mantendo dois deles fixos e modificando um deles para

podermos analisar o seu impacto da otimização das rotas através do feromônio e o tempo de execução do algoritmo no gráfico gerado para cada uma das variações.

```
Distancias:
   | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 0| 0| 3| 9| 7|13| 6| 6|17|14|13|17| 3| 5|14|16|
 1| 3| 0|19|12| 6| 1| 5| 4|14|13|20|18|13| 7|12|
 2| 9|19| 0| 2|10| 6|14|18|13| 6| 7| 7|15| 2|19|
 3| 7|12| 2| 0|12| 9|11|19|11| 3|17|11|10|13|17|
 4|13| 6|10|12| 0|17| 3| 3| 9|11| 5|11|15| 4| 7|
 5| 6| 1| 6| 9|17| 0|16| 9|14|15| 3|14| 1| 8| 5|
 6| 6| 5|14|11| 3|16| 0|20| 4|12|15| 2| 8| 9|17|
 7|17| 4|18|19| 3| 9|20| 0| 4| 5|10|19| 5|10|11|
 8|14|14|13|11| 9|14| 4| 4| 0|15|13|10|15| 1| 1|
 9|13|13| 6| 3|11|15|12| 5|15| 0| 5|19| 1| 4|12|
10|17|20| 7|17| 5| 3|15|10|13| 5| 0|11| 5|20|16|
11| 3|18| 7|11|11|14| 2|19|10|19|11| 0|19| 7| 7|
12| 5|13|15|10|15| 1| 8| 5|15| 1| 5|19| 0|15| 1|
13|14| 7| 2|13| 4| 8| 9|10| 1| 4|20| 7|15| 0| 6|
14|16|12|19|17| 7| 5|17|11| 1|12|16| 7| 1| 6| 0|
---+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

Início da viagem 0

Formiga  0 fez a rota: [4, 7, 5, 8, 3, 1, 14, 11, 12, 9, 13, 6, 0, 10, 2, 4] de distancia 141
Formiga  1 fez a rota: [13, 9, 3, 2, 6, 1, 8, 14, 4, 12, 5, 11, 7, 0, 10, 13] de distancia 153
Formiga  2 fez a rota: [7, 8, 11, 9, 4, 3, 12, 10, 6, 14, 2, 0, 5, 1, 13, 7] de distancia 155
Formiga  3 fez a rota: [5, 7, 1, 8, 9, 11, 10, 3, 2, 0, 14, 13, 6, 12, 4, 5] de distancia 171
Formiga  4 fez a rota: [1, 10, 9, 5, 13, 4, 6, 0, 14, 11, 2, 7, 3, 8, 12, 1] de distancia 167
Formiga  5 fez a rota: [8, 0, 10, 14, 7, 13, 5, 9, 6, 12, 1, 3, 11, 2, 4, 8] de distancia 173
Formiga  6 fez a rota: [3, 14, 6, 2, 13, 11, 0, 8, 1, 7, 9, 10, 5, 4, 12, 3] de distancia 147
Formiga  7 fez a rota: [5, 0, 7, 14, 12, 2, 8, 6, 9, 11, 3, 1, 10, 4, 13, 5] de distancia 158
Formiga  8 fez a rota: [6, 0, 14, 1, 11, 9, 5, 3, 12, 8, 4, 2, 10, 7, 13, 6] de distancia 175
Formiga  9 fez a rota: [2, 3, 8, 4, 5, 10, 9, 13, 1, 12, 0, 14, 7, 11, 6, 2] de distancia 138
Formiga 10 fez a rota: [2, 0, 11, 3, 7, 14, 9, 8, 4, 10, 13, 12, 1, 5, 6, 2] de distancia 173

melhor rota: [2, 3, 8, 4, 5, 10, 9, 13, 1, 12, 0, 14, 7, 11, 6, 2] de distancia 138
```

```
Feromonios:
  |     0|     1|     2|     3|     4|     5|     6|     7|     8|     9|    10|    11|    12|    13|    14|
--+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+
 0|0,0100|0,0090|0,0433|0,0090|0,0090|0,0331|0,0443|0,0332|0,0330|0,0090|0,0447|0,0336|0,0233|0,0090|0,0567|
 1|0,0090|0,0100|0,0090|0,0451|0,0090|0,0324|0,0209|0,0329|0,0449|0,0090|0,0325|0,0202|0,0459|0,0352|0,0330|
 2|0,0433|0,0090|0,0100|0,0461|0,0311|0,0090|0,0340|0,0202|0,0212|0,0090|0,0330|0,0312|0,0212|0,0220|0,0209|
 3|0,0090|0,0451|0,0461|0,0100|0,0209|0,0202|0,0090|0,0318|0,0474|0,0209|0,0199|0,0438|0,0321|0,0090|0,0220|
 4|0,0090|0,0090|0,0311|0,0209|0,0100|0,0364|0,0202|0,0218|0,0461|0,0209|0,0328|0,0090|0,0449|0,0325|0,0209|
 5|0,0331|0,0324|0,0090|0,0202|0,0364|0,0100|0,0205|0,0327|0,0218|0,0424|0,0364|0,0209|0,0209|0,0312|0,0090|
 6|0,0443|0,0209|0,0340|0,0090|0,0202|0,0205|0,0100|0,0090|0,0212|0,0322|0,0209|0,0233|0,0308|0,0327|0,0339|
 7|0,0332|0,0329|0,0202|0,0318|0,0218|0,0327|0,0090|0,0100|0,0209|0,0220|0,0202|0,0352|0,0090|0,0311|0,0581|
 8|0,0330|0,0449|0,0212|0,0474|0,0461|0,0218|0,0212|0,0209|0,0100|0,0314|0,0090|0,0209|0,0314|0,0090|0,0209|
 9|0,0090|0,0090|0,0090|0,0209|0,0209|0,0424|0,0322|0,0220|0,0314|0,0100|0,0476|0,0552|0,0218|0,0481|0,0205|
10|0,0447|0,0325|0,0330|0,0199|0,0328|0,0364|0,0209|0,0202|0,0090|0,0476|0,0100|0,0199|0,0209|0,0205|0,0199|
11|0,0336|0,0202|0,0312|0,0438|0,0090|0,0209|0,0233|0,0352|0,0209|0,0552|0,0199|0,0100|0,0218|0,0220|0,0331|
12|0,0233|0,0459|0,0212|0,0321|0,0449|0,0209|0,0308|0,0090|0,0314|0,0218|0,0209|0,0218|0,0100|0,0205|0,0212|
13|0,0090|0,0352|0,0220|0,0090|0,0325|0,0312|0,0327|0,0311|0,0090|0,0481|0,0205|0,0220|0,0205|0,0100|0,0199|
14|0,0567|0,0330|0,0209|0,0220|0,0209|0,0090|0,0339|0,0581|0,0209|0,0205|0,0199|0,0331|0,0212|0,0199|0,0100|
--+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+

Fim da viagem 0


Início da viagem 1

Formiga  0 fez a rota: [0, 2, 10, 5, 1, 7, 8, 14, 12, 4, 13, 9, 11, 6, 3, 0] de distancia 92
Formiga  1 fez a rota: [8, 14, 7, 13, 9, 10, 12, 6, 0, 5, 1, 3, 2, 4, 11, 8] de distancia 102
Formiga  2 fez a rota: [10, 5, 1, 8, 3, 2, 13, 4, 12, 9, 7, 14, 11, 0, 6, 10] de distancia 100
Formiga  3 fez a rota: [2, 13, 9, 12, 14, 7, 1, 5, 10, 0, 11, 6, 4, 8, 3, 2] de distancia 74
Formiga  4 fez a rota: [13, 9, 12, 14, 8, 4, 10, 5, 1, 7, 11, 6, 0, 2, 3, 13] de distancia 80
Formiga  5 fez a rota: [2, 3, 9, 13, 4, 8, 14, 0, 5, 1, 7, 11, 6, 12, 10, 2] de distancia 91
Formiga  6 fez a rota: [10, 5, 1, 6, 0, 14, 12, 9, 13, 8, 4, 7, 11, 3, 2, 10] de distancia 89
Formiga  7 fez a rota: [8, 6, 11, 2, 3, 12, 1, 5, 10, 9, 13, 4, 7, 14, 0, 8] de distancia 99
Formiga  8 fez a rota: [7, 5, 1, 3, 2, 13, 4, 6, 0, 11, 14, 12, 9, 10, 8, 7] de distancia 73
Formiga  9 fez a rota: [8, 14, 7, 1, 5, 12, 9, 10, 4, 13, 6, 0, 11, 3, 2, 8] de distancia 77
Formiga 10 fez a rota: [14, 12, 5, 1, 3, 2, 10, 9, 11, 6, 4, 8, 13, 7, 0, 14] de distancia 106

melhor rota: [7, 5, 1, 3, 2, 13, 4, 6, 0, 11, 14, 12, 9, 10, 8, 7] de distancia 73
```

```
Feromonios:
  |     0|     1|     2|     3|     4|     5|     6|     7|     8|     9|    10|    11|    12|    13|    14|
--+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+
 0|0,0100|0,0080|0,0732|0,0080|0,0080|0,0539|0,0749|0,0539|0,0536|0,0080|0,0758|0,0547|0,0352|0,0080|0,0985|
 1|0,0080|0,0100|0,0080|0,0764|0,0080|0,0525|0,0306|0,0534|0,0761|0,0080|0,0527|0,0292|0,0780|0,0578|0,0536|
 2|0,0732|0,0080|0,0100|0,0785|0,0501|0,0080|0,0554|0,0294|0,0313|0,0080|0,0536|0,0502|0,0313|0,0328|0,0306|
 3|0,0080|0,0764|0,0785|0,0100|0,0306|0,0292|0,0080|0,0513|0,0809|0,0306|0,0287|0,0740|0,0518|0,0080|0,0328|
 4|0,0080|0,0080|0,0501|0,0306|0,0100|0,0600|0,0294|0,0323|0,0784|0,0306|0,0532|0,0080|0,0761|0,0527|0,0306|
 5|0,0539|0,0525|0,0080|0,0292|0,0600|0,0100|0,0299|0,0530|0,0323|0,0715|0,0600|0,0306|0,0306|0,0502|0,0080|
 6|0,0749|0,0306|0,0554|0,0080|0,0294|0,0299|0,0100|0,0080|0,0313|0,0521|0,0306|0,0352|0,0495|0,0530|0,0554|
 7|0,0539|0,0534|0,0294|0,0513|0,0323|0,0530|0,0080|0,0100|0,0306|0,0328|0,0292|0,0578|0,0080|0,0501|0,1012|
 8|0,0536|0,0761|0,0313|0,0809|0,0784|0,0323|0,0313|0,0306|0,0100|0,0506|0,0080|0,0306|0,0506|0,0080|0,0306|
 9|0,0080|0,0080|0,0080|0,0306|0,0306|0,0715|0,0521|0,0328|0,0506|0,0100|0,0814|0,0958|0,0323|0,0822|0,0299|
10|0,0758|0,0527|0,0536|0,0287|0,0532|0,0600|0,0306|0,0292|0,0080|0,0814|0,0100|0,0287|0,0306|0,0299|0,0289|
11|0,0547|0,0292|0,0502|0,0740|0,0080|0,0306|0,0352|0,0578|0,0306|0,0958|0,0287|0,0100|0,0323|0,0328|0,0537|
12|0,0352|0,0780|0,0313|0,0518|0,0761|0,0306|0,0495|0,0080|0,0506|0,0323|0,0306|0,0323|0,0100|0,0299|0,0313|
13|0,0080|0,0578|0,0328|0,0080|0,0527|0,0502|0,0530|0,0501|0,0080|0,0822|0,0299|0,0328|0,0299|0,0100|0,0287|
14|0,0985|0,0536|0,0306|0,0328|0,0306|0,0080|0,0554|0,1012|0,0306|0,0299|0,0289|0,0537|0,0313|0,0287|0,0100|
--+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+

Fim da viagem 1
```

```
Inicio da viagem 2

Formiga  0 fez a rota: [5, 1, 7, 4, 13, 9, 12, 14, 8, 3, 2, 10, 0, 11, 6, 5] de distancia 77
Formiga  1 fez a rota: [5, 1, 7, 14, 8, 4, 6, 0, 11, 2, 3, 9, 12, 10, 13, 5] de distancia 84
Formiga  2 fez a rota: [2, 3, 11, 6, 4, 13, 9, 12, 14, 8, 1, 5, 10, 0, 7, 2] de distancia 99
Formiga  3 fez a rota: [13, 9, 12, 5, 1, 7, 14, 8, 4, 6, 11, 2, 3, 10, 0, 13] de distancia 94
Formiga  4 fez a rota: [2, 3, 8, 14, 12, 9, 13, 4, 7, 1, 5, 10, 0, 6, 11, 2] de distancia 67
Formiga  5 fez a rota: [7, 14, 12, 9, 10, 5, 1, 3, 2, 13, 4, 8, 6, 11, 0, 7] de distancia 77
Formiga  6 fez a rota: [3, 2, 13, 9, 12, 14, 8, 4, 7, 1, 5, 10, 0, 11, 6, 3] de distancia 64
Formiga  7 fez a rota: [12, 14, 8, 4, 13, 9, 10, 5, 1, 7, 11, 6, 0, 2, 3, 12] de distancia 80
Formiga  8 fez a rota: [3, 2, 6, 11, 0, 14, 12, 9, 13, 8, 4, 5, 1, 7, 10, 3] de distancia 102
Formiga  9 fez a rota: [12, 9, 10, 5, 1, 7, 14, 8, 4, 6, 11, 0, 2, 3, 13, 12] de distancia 82
Formiga 10 fez a rota: [14, 8, 4, 7, 1, 5, 10, 12, 9, 13, 2, 3, 11, 6, 0, 14] de distancia 70

melhor rota: [3, 2, 13, 9, 12, 14, 8, 4, 7, 1, 5, 10, 0, 11, 6, 3] de distancia 64

Feromonios:
  |     0|     1|     2|     3|     4|     5|     6|     7|     8|     9|    10|    11|    12|    13|    14|
--+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+
 0|0,0100|0,0072|0,1087|0,0072|0,0072|0,0870|0,2011|0,0672|0,0480|0,0072|0,0931|0,1452|0,0315|0,0072|0,1503|
 1|0,0072|0,0100|0,0072|0,1321|0,0072|0,2853|0,0490|0,1629|0,0866|0,0072|0,0472|0,0262|0,0894|0,0517|0,0480|
 2|0,1087|0,0072|0,0100|0,2637|0,0628|0,0072|0,0496|0,0263|0,0280|0,0072|0,0865|0,0646|0,0280|0,0999|0,0274|
 3|0,0072|0,1321|0,2637|0,0100|0,0274|0,0262|0,0268|0,0459|0,1161|0,0483|0,0257|0,1136|0,0660|0,0072|0,0294|
 4|0,0072|0,0072|0,0628|0,0274|0,0100|0,0537|0,0972|0,0701|0,1802|0,0274|0,0969|0,0251|0,1062|0,1783|0,0274|
 5|0,0870|0,2853|0,0072|0,0262|0,0537|0,0100|0,0268|0,0743|0,0289|0,0640|0,1817|0,0274|0,0720|0,0450|0,0072|
 6|0,2011|0,0490|0,0496|0,0268|0,0972|0,0268|0,0100|0,0072|0,0476|0,0467|0,0274|0,1593|0,0831|0,0732|0,0496|
 7|0,0672|0,1629|0,0263|0,0459|0,0701|0,0743|0,0072|0,0100|0,0470|0,0478|0,0262|0,1178|0,0072|0,0816|0,1975|
 8|0,0480|0,0866|0,0280|0,1161|0,1802|0,0289|0,0476|0,0470|0,0100|0,0453|0,0340|0,0274|0,0453|0,0476|0,1352|
 9|0,0072|0,0072|0,0072|0,0483|0,0274|0,0640|0,0467|0,0478|0,0453|0,0100|0,1818|0,1243|0,1703|0,2219|0,0268|
10|0,0931|0,0472|0,0865|0,0257|0,0969|0,1817|0,0274|0,0262|0,0340|0,1818|0,0100|0,0257|0,0662|0,0268|0,0258|
11|0,1452|0,0262|0,0646|0,1136|0,0251|0,0274|0,1593|0,1178|0,0274|0,1243|0,0257|0,0100|0,0289|0,0294|0,0934|
12|0,0315|0,0894|0,0280|0,0660|0,1062|0,0720|0,0831|0,0072|0,0453|0,1703|0,0662|0,0289|0,0100|0,0268|0,1637|
13|0,0072|0,0517|0,0999|0,0072|0,1783|0,0450|0,0732|0,0816|0,0476|0,2219|0,0268|0,0294|0,0268|0,0100|0,0257|
14|0,1503|0,0480|0,0274|0,0294|0,0274|0,0072|0,0496|0,1975|0,1352|0,0268|0,0258|0,0934|0,1637|0,0257|0,0100|
--+------+------+------+------+------+------+------+------+------+------+------+------+------+------+------+

Fim da viagem 2
```

Requisitos necessários para reproduzir resultados

possuir o jdk instalado no sistema com as variáveis de de ambiente configuradas
sendo elas

nome	caminho
JAVA_HOME	ex:C:\Program Files\Java\jdk-17.0.2
path	ex:C:\Program Files\Java\jdk-17.0.2/bin

Exemplo de como executar experimentos.

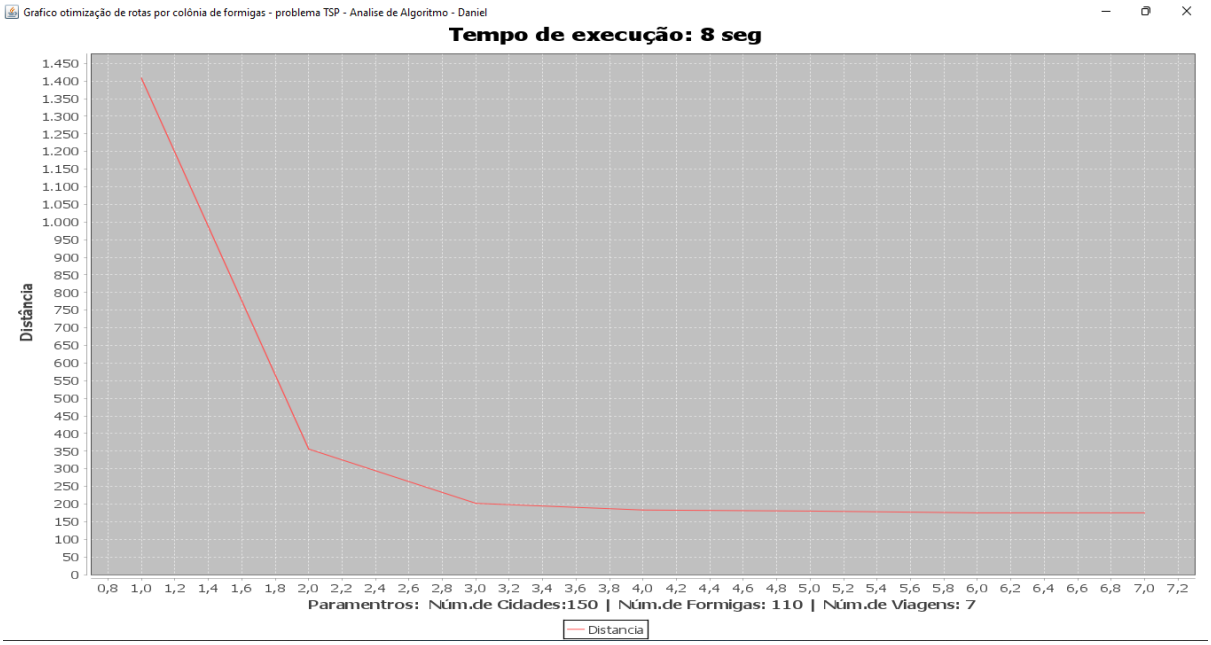
usando o comando no prompt javac "local do arquivo.java" para compilar o código e depois java “local do arquivo.class”

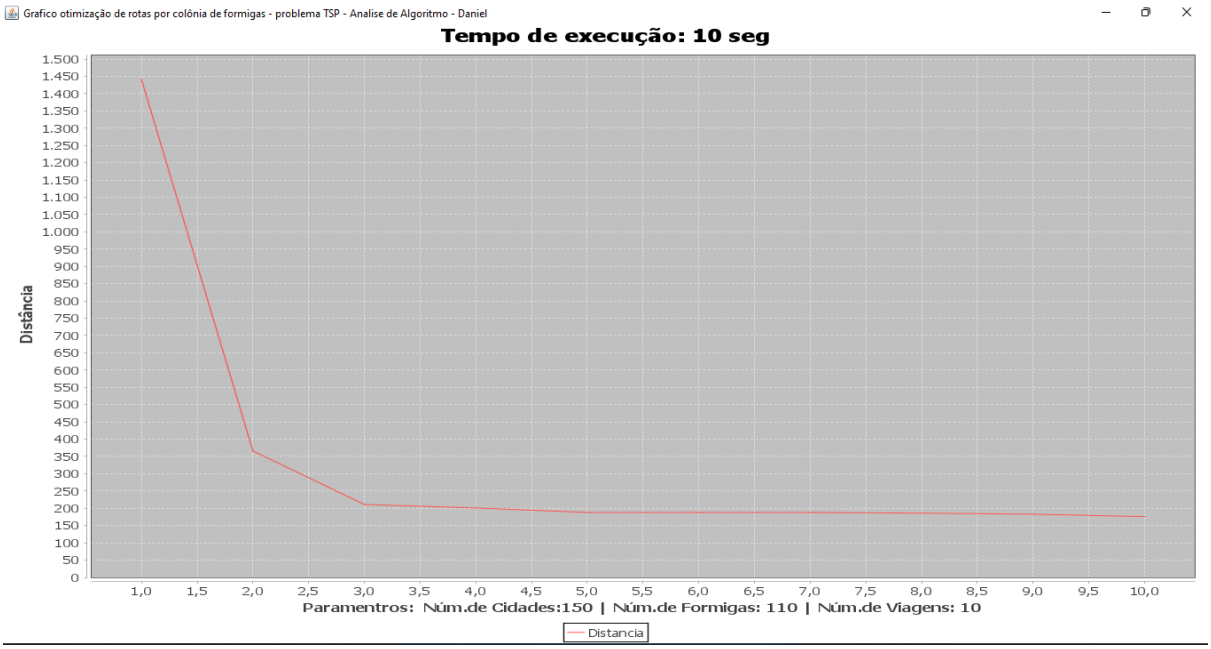
Resultados da Avaliação

Nesta seção vamos analisar os resultados do impacto no aumento e redução das variáveis com gráficos gerados apresentando os resultados.

Resultados do 1º experimento aumentando o número de viagens

neste primeiro experimento como mencionado anteriormente manterei dois valores padrões e começarei aumentando gradativamente o número de viagens





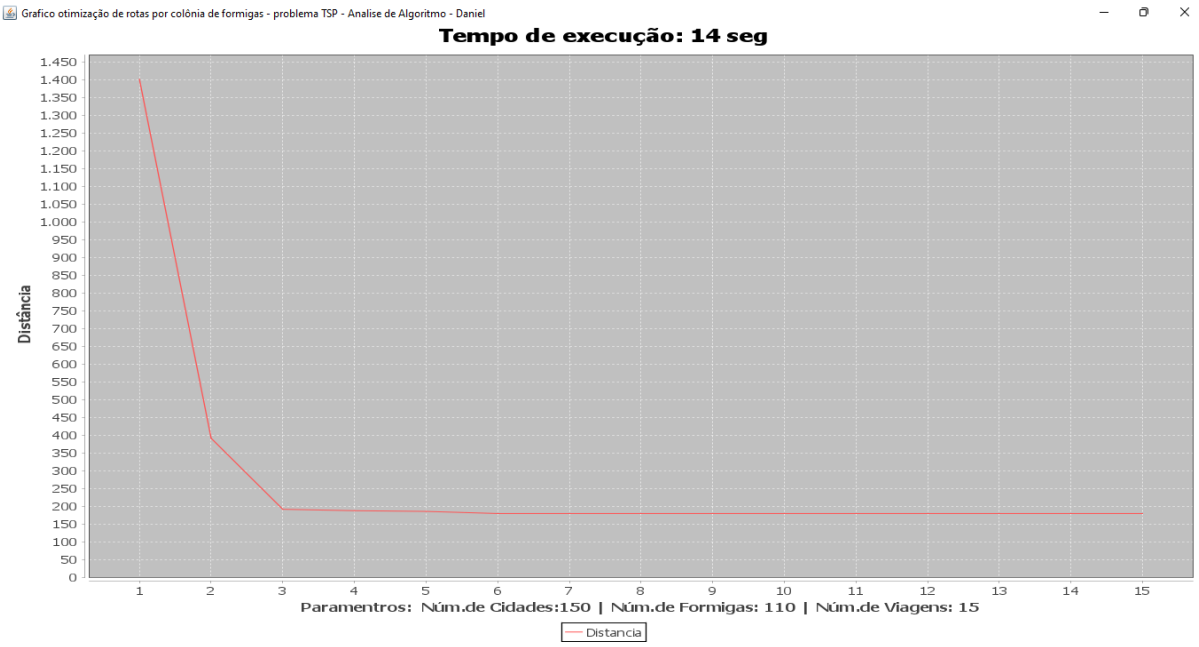


Grafico otimização de rotas por colônia de formigas - problema TSP - Analise de Algoritmo - Daniel
Tempo de execução: 14 seg
Distância
Paramentros: Núm.de Cidades:150 | Núm.de Formigas: 110 | Núm.de Viagens: 15
Distancia

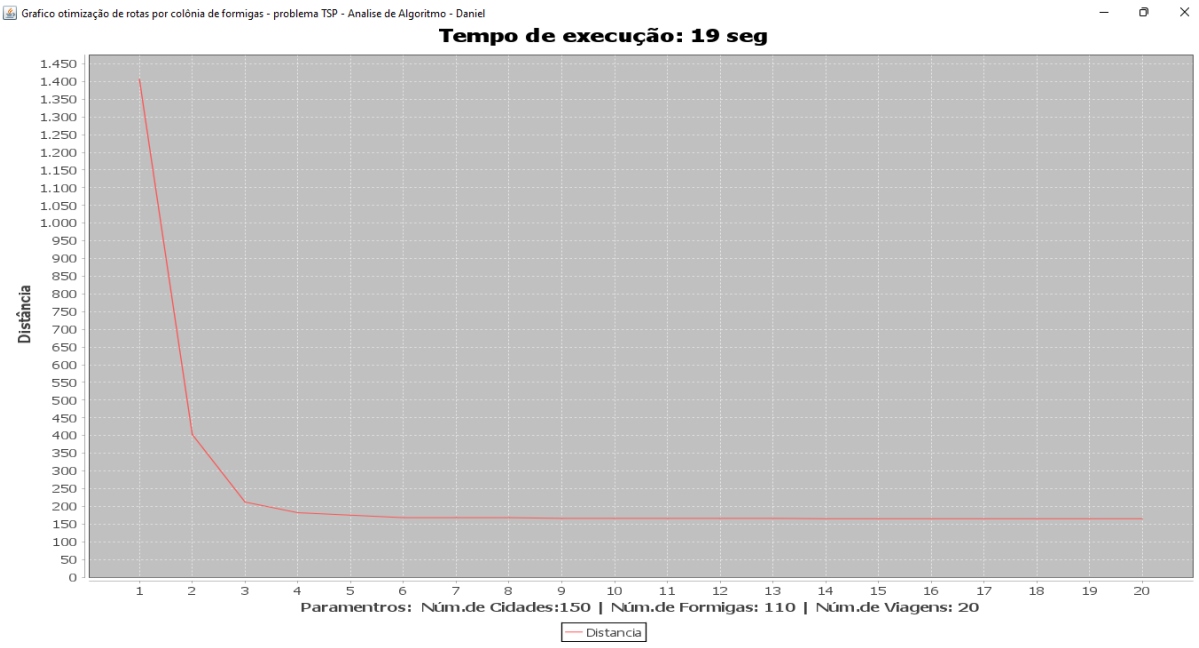
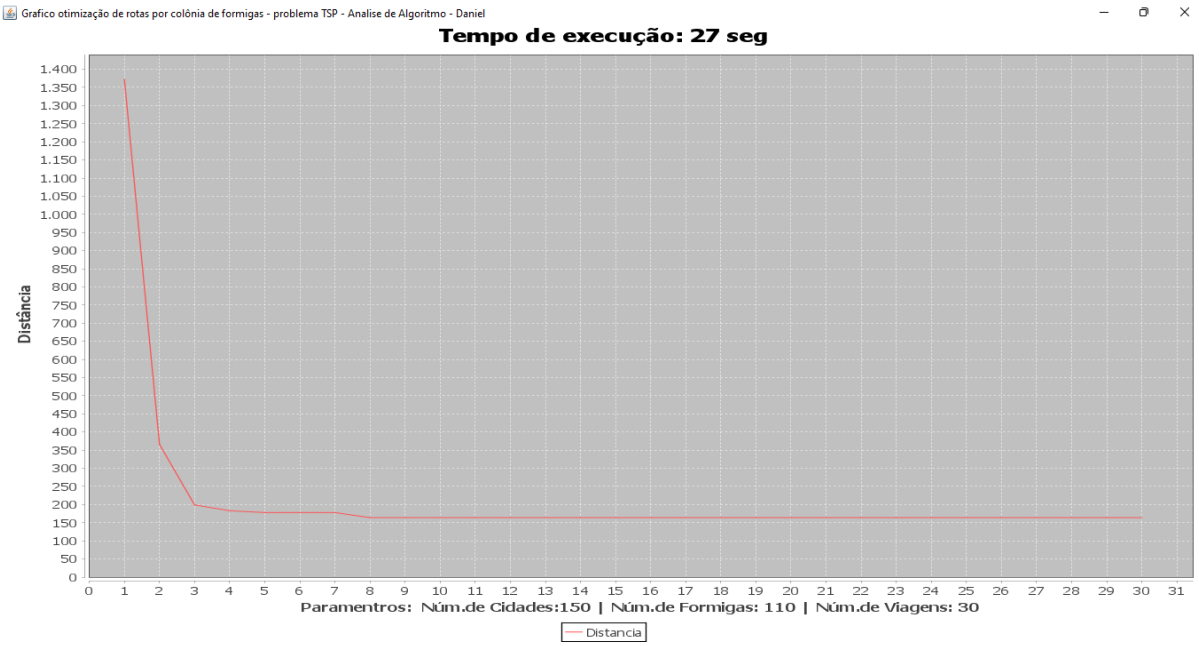
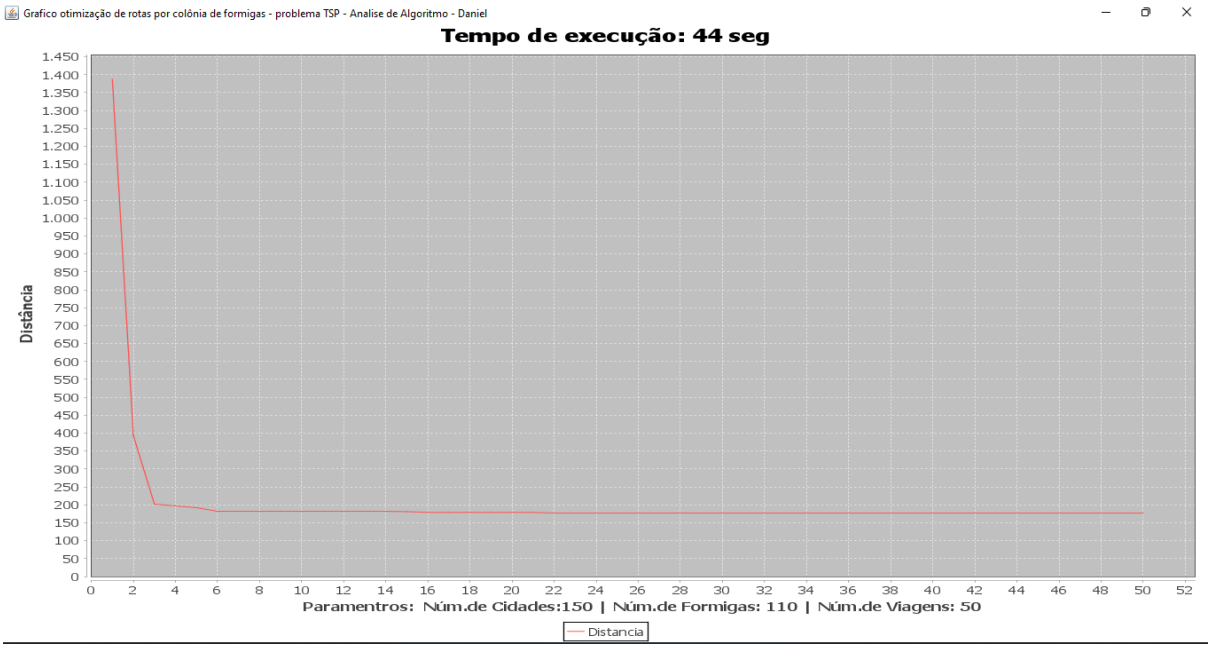


Grafico otimização de rotas por colônia de formigas - problema TSP - Analise de Algoritmo - Daniel
Tempo de execução: 19 seg
Distância
Paramentros: Núm.de Cidades:150 | Núm.de Formigas: 110 | Núm.de Viagens: 20
Distancia

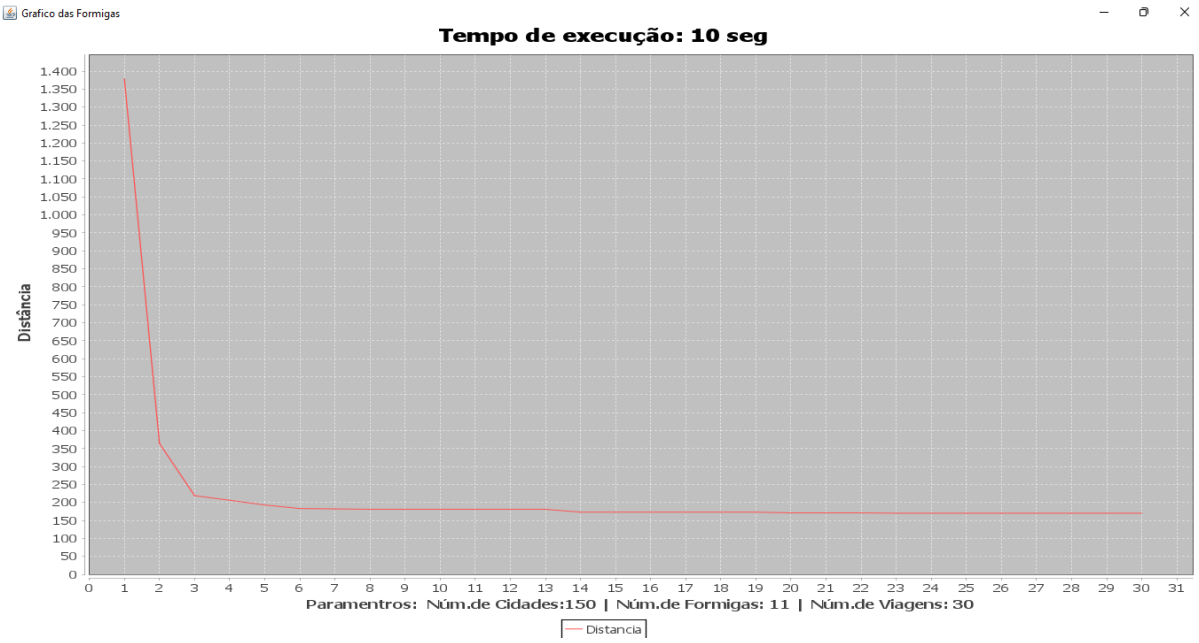


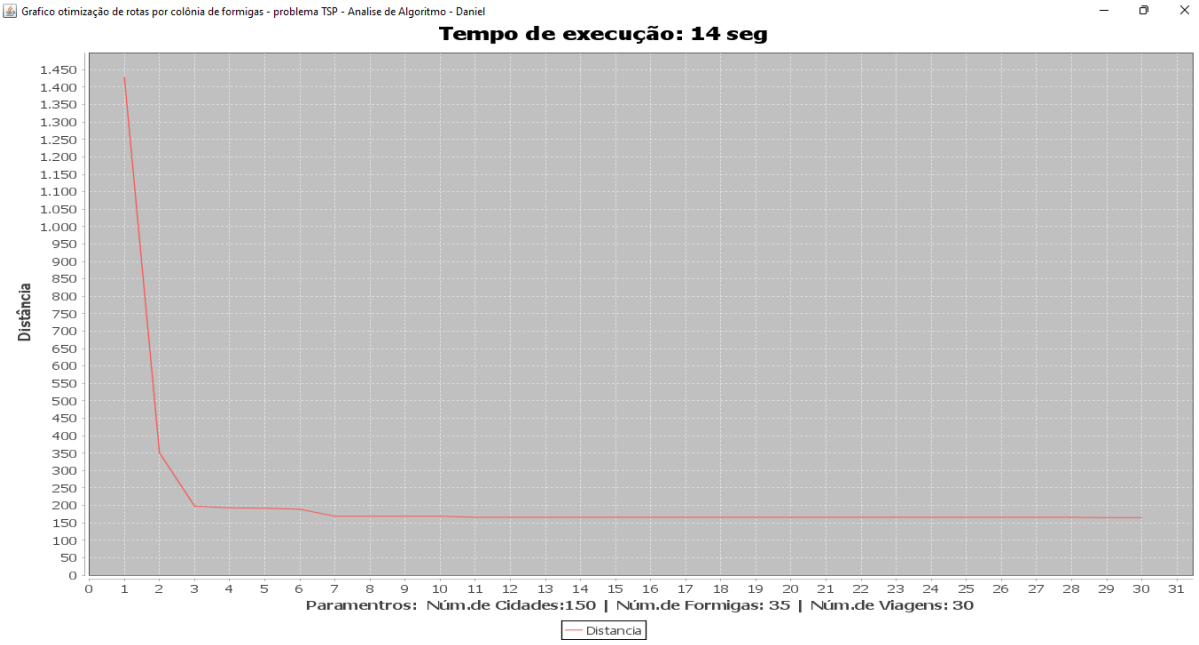


Ao analisarmos os gráficos obtidos com o aumento do parâmetro número de viagens podemos concluir que o tempo de execução aumenta conforme o número de viagens sendo apenas significante após 50 viagens.

Resultado do 2° experimento aumento no número de formigas

No parâmetro da formiga, utilizei valores de 11, 35, 65. 85, 110, 200. A partir da avaliação destes experimentos





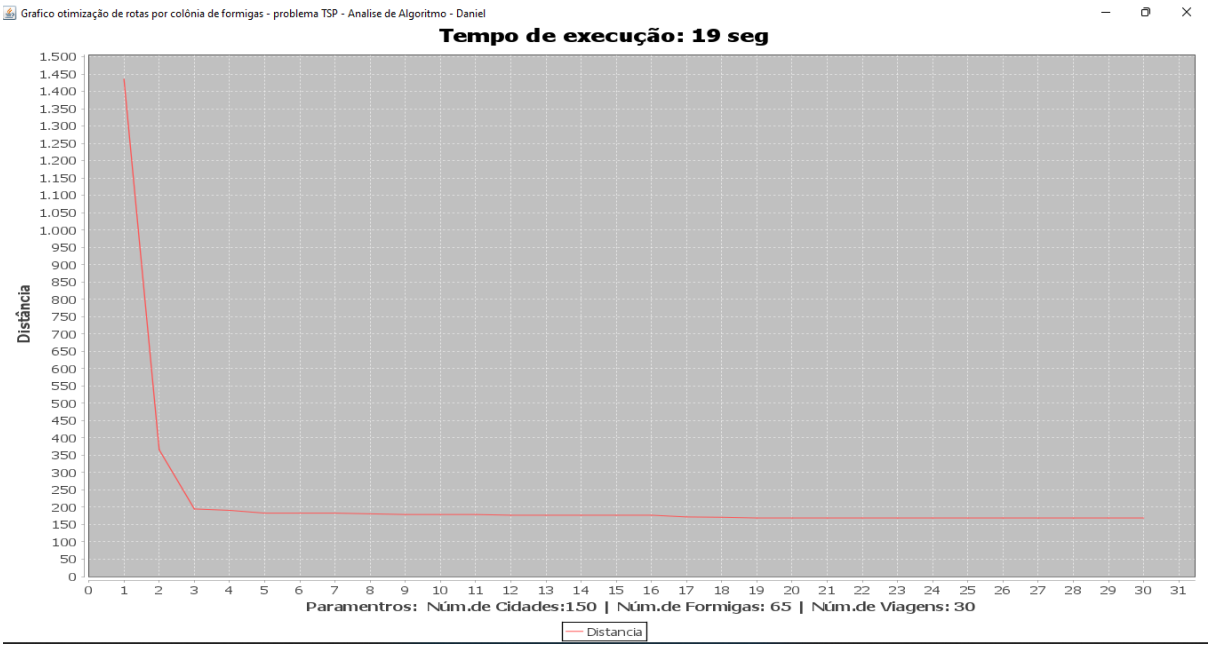


Grafico otimização de rotas por colônia de formigas - problema TSP - Analise de Algoritmo - Daniel
Tempo de execução: 19 seg
Distância
Paramentros: Núm.de Cidades:150 | Núm.de Formigas: 65 | Núm.de Viagens: 30
Distancia

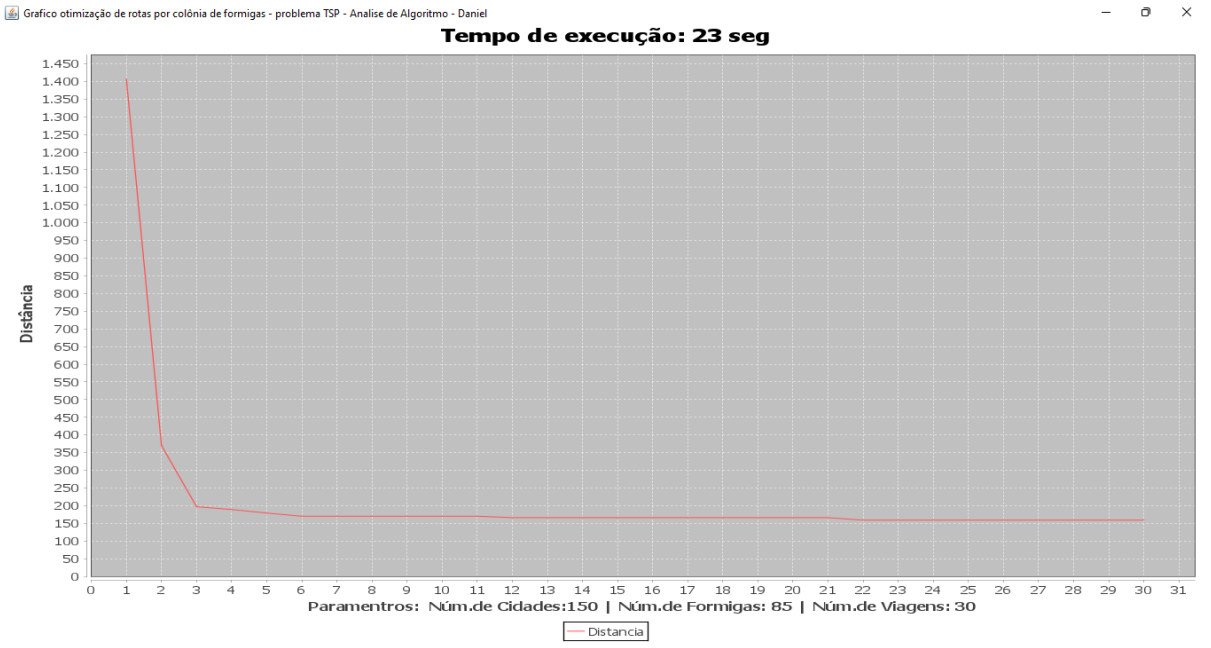
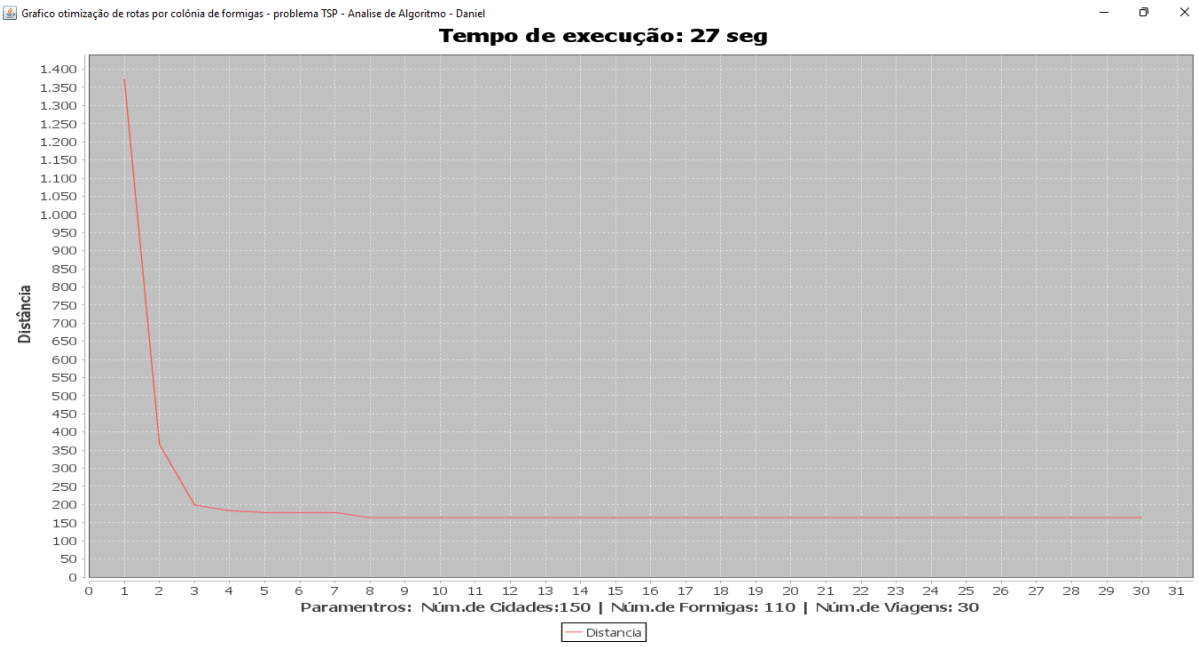
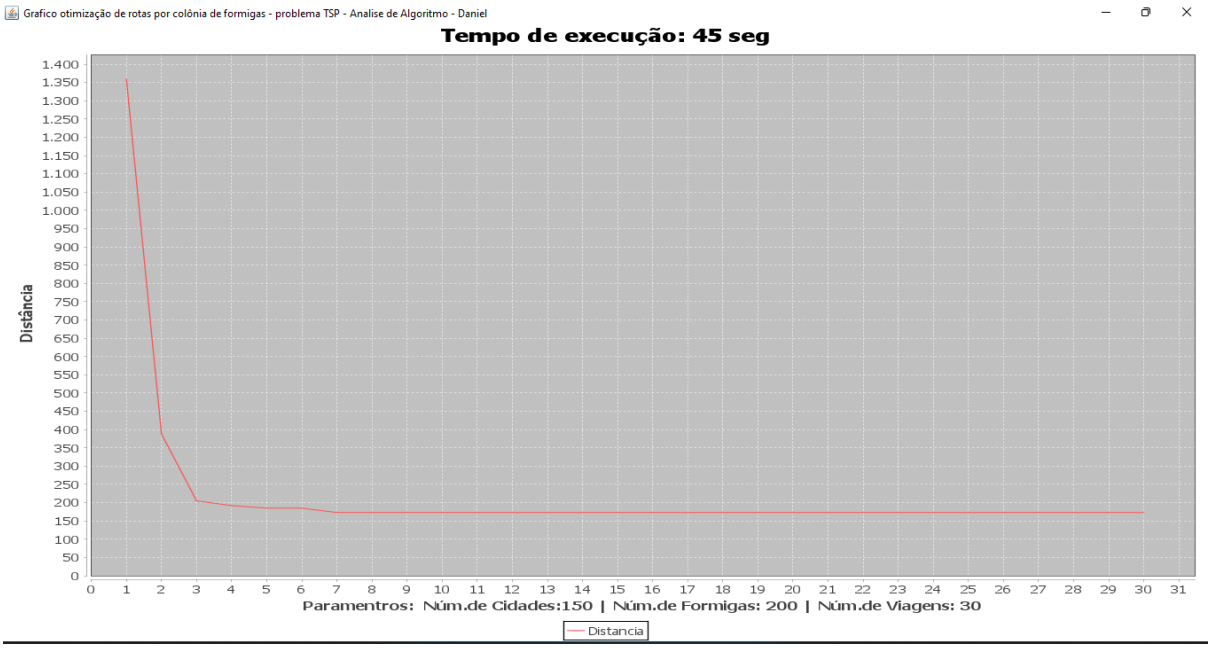


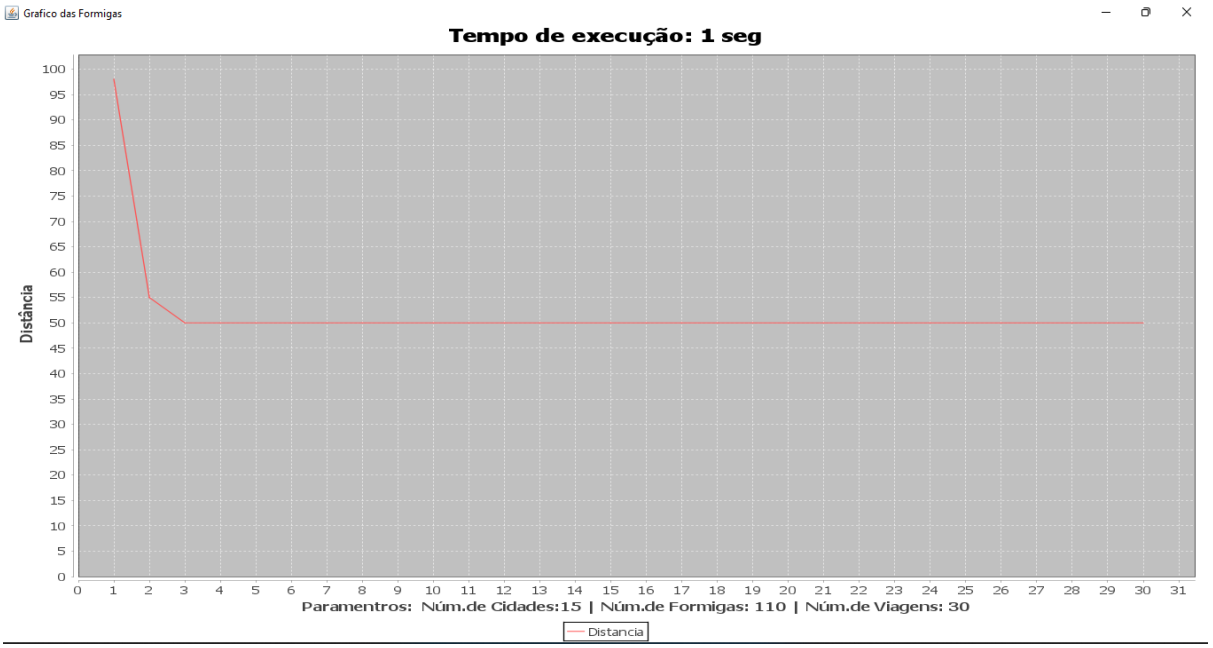
Grafico otimização de rotas por colônia de formigas - problema TSP - Analise de Algoritmo - Daniel
Tempo de execução: 23 seg
Distância
Paramentros: Núm.de Cidades:150 | Núm.de Formigas: 85 | Núm.de Viagens: 30
Distancia

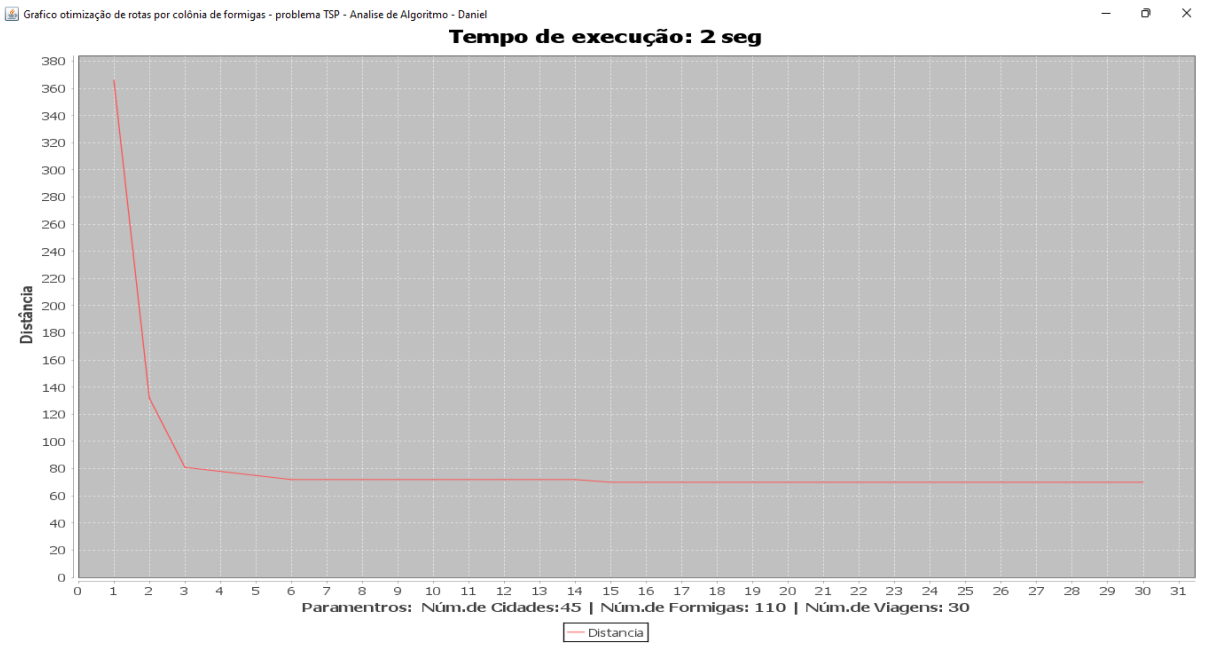




A partir da avaliação experimental com os parâmetros propostos em uma aumento da população de formigas podemos analisar a otimização da distância tendo pouco impacto no tempo de execução do algoritmo

Resultado do 3° experimento aumento no número de cidades





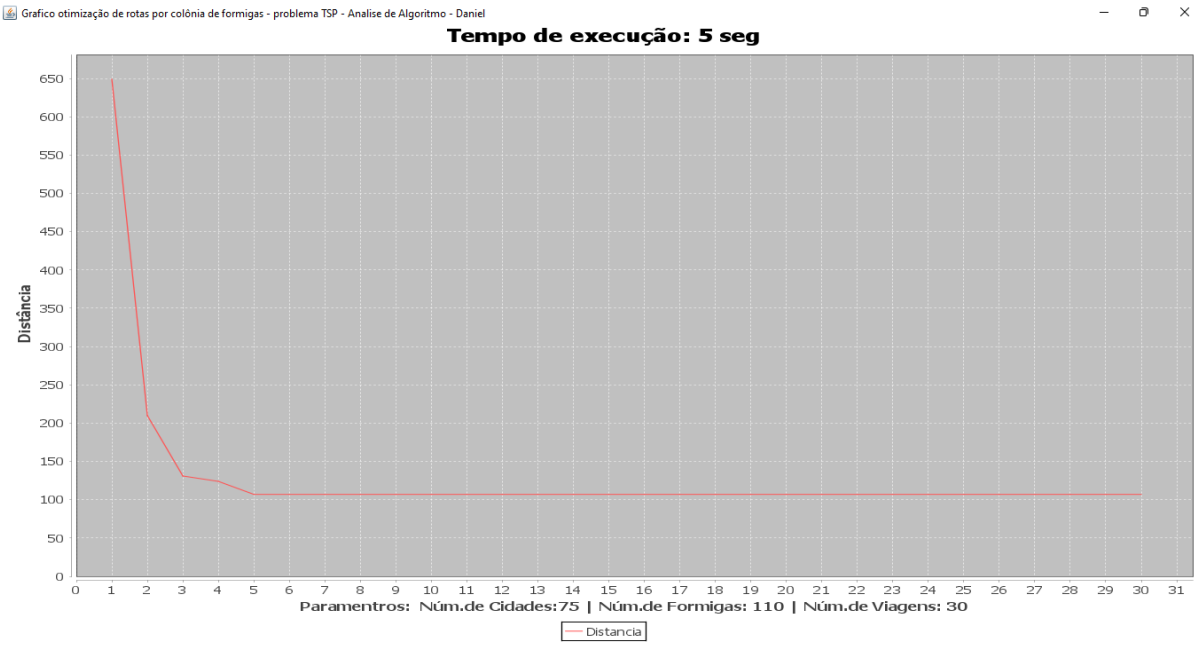


Grafico otimização de rotas por colônia de formigas - problema TSP - Analise de Algoritmo - Daniel
Tempo de execução: 5 seg
Distância
Paramentros: Núm.de Cidades:75 | Núm.de Formigas: 110 | Núm.de Viagens: 30
Distancia

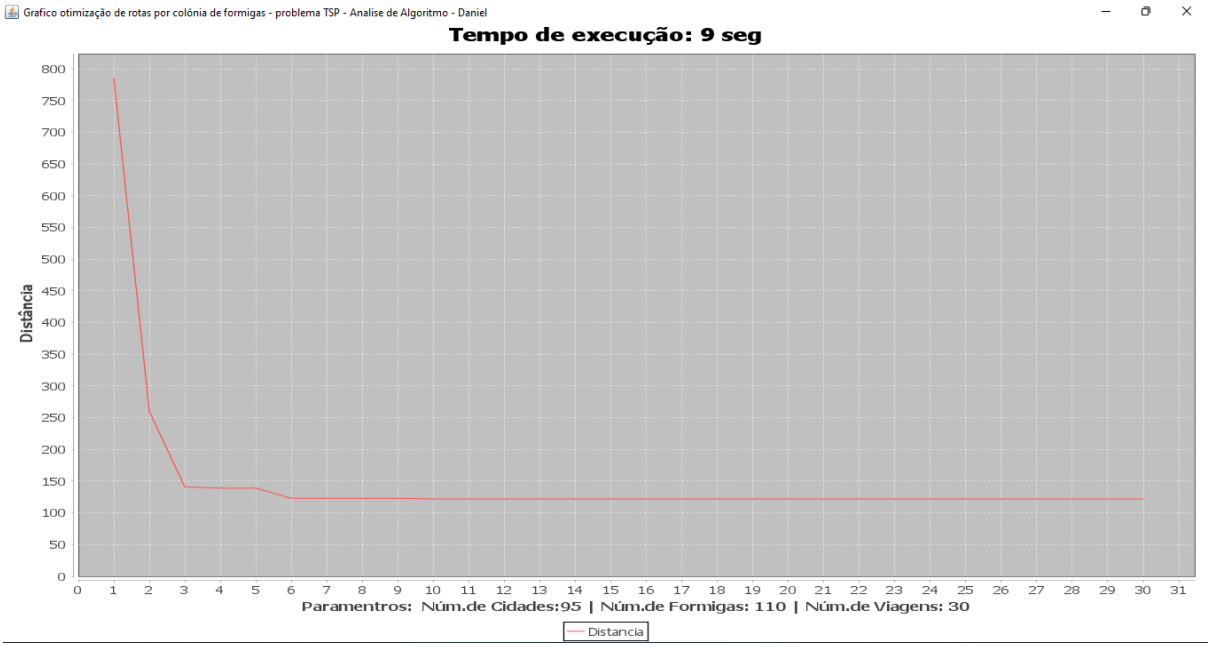
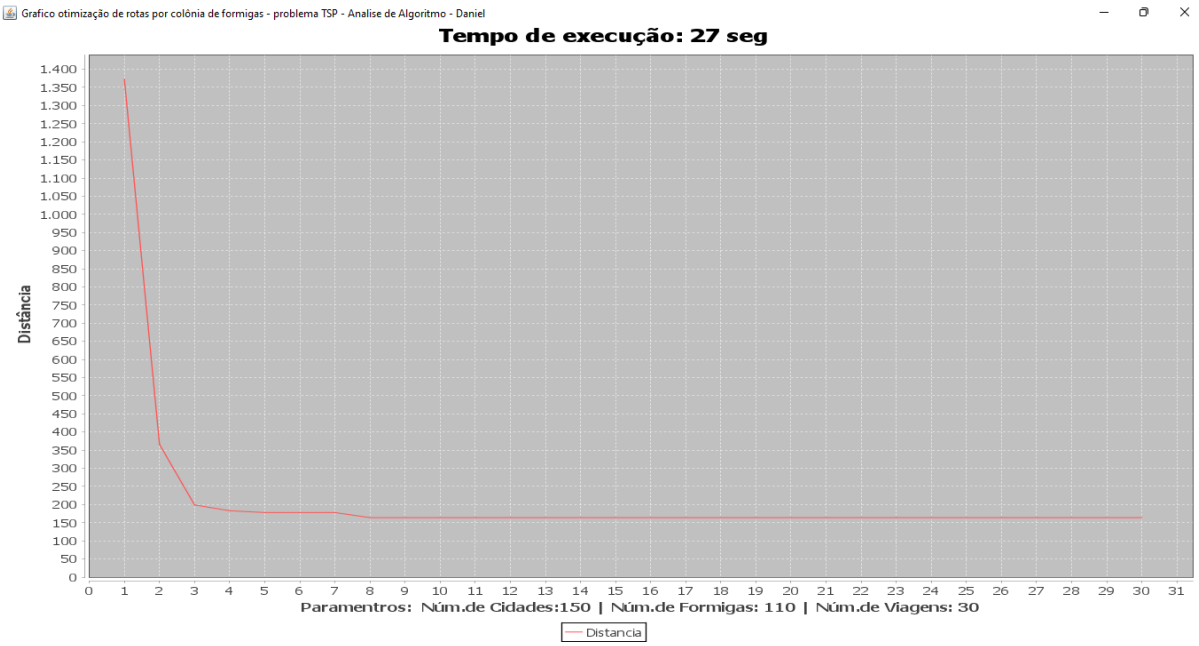
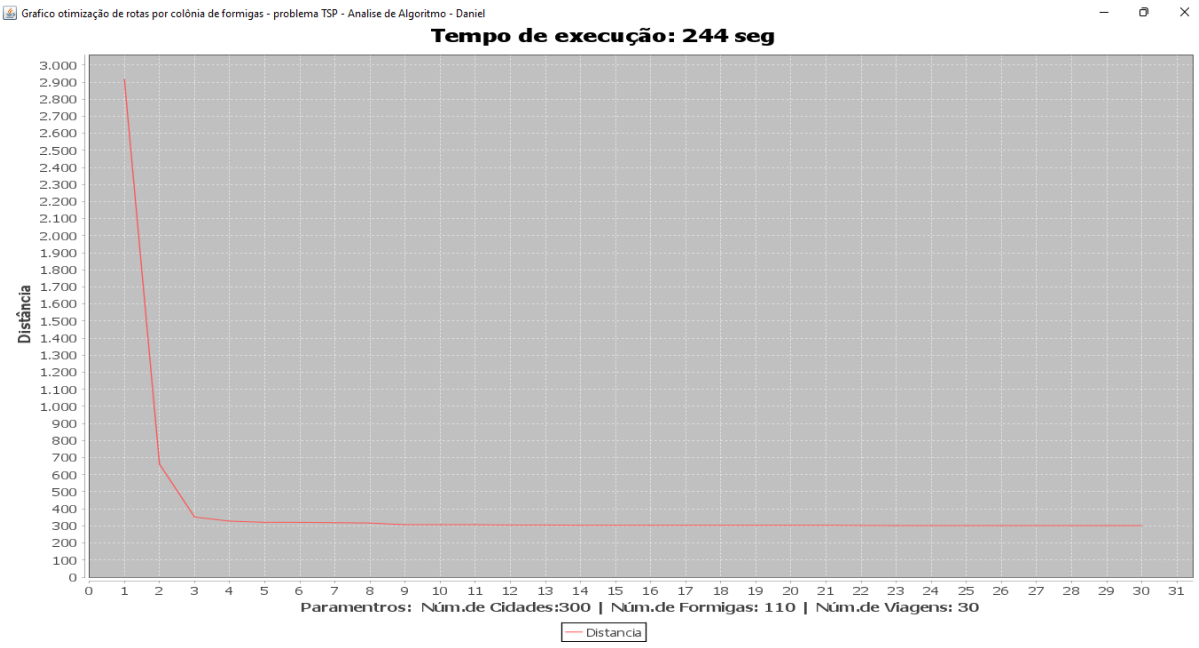
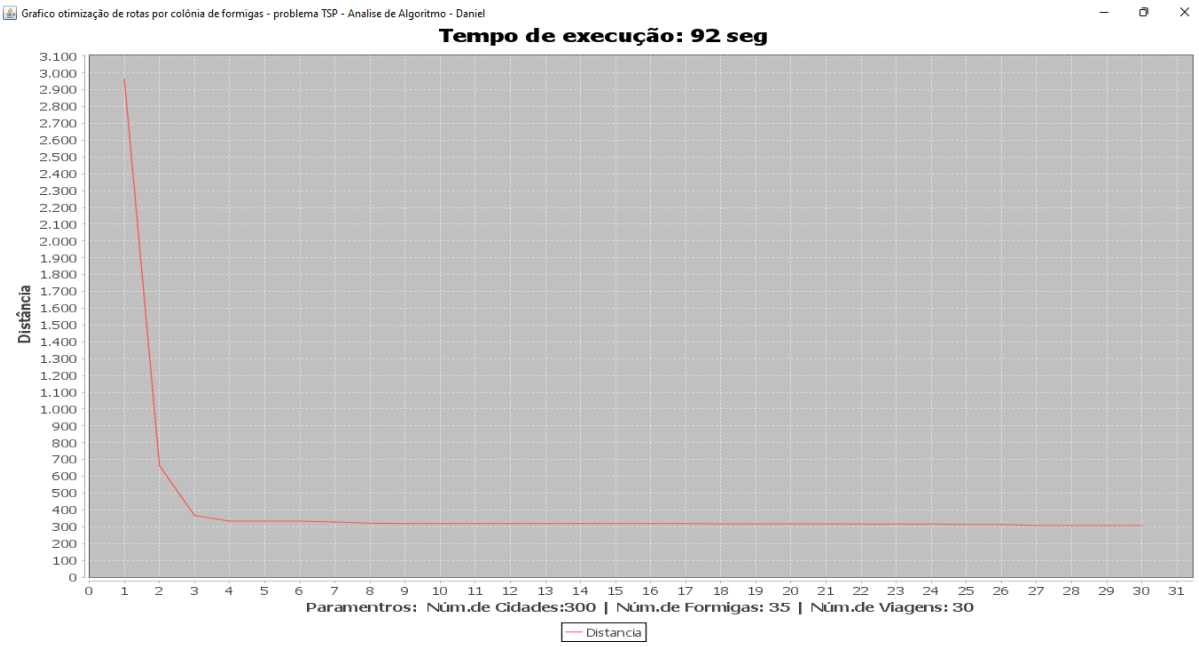


Grafico otimização de rotas por colônia de formigas - problema TSP - Analise de Algoritmo - Daniel
Tempo de execução: 9 seg
Distância
Paramentros: Núm.de Cidades:95 | Núm.de Formigas: 110 | Núm.de Viagens: 30
Distancia





Ao analisarmos os gráficos obtidos podemos concluir que o tempo de execução do experimento é aumentado drasticamente quando passa de 300 e comparado com o número de cidades abaixo de 150 outro fato que me deparei no experimento foi que ao reduzirmos o número de formigas e mantendo os outros valores temos uma redução no tempo de execução do algoritmo como podemos ver na figura abaixo



Conclusão dos experimentos

comparando os resultados dos gráficos obtidos com a variação dos três diferentes parâmetros podemos verificar que ao aumentarmos o número de cidades e diminuir o número de formigas o tempo de execução tende a diminuir diferentemente dos outros casos que o tempo de execução aumenta gradativamente tendo impacto no algoritmo quando os números ultrapassam 200.

Projeto da avaliação Analitica

Para realizar a avaliação analítica usei o problema do caixeiro viajante usando o algoritmo colônia de formigas para otimização das rotas compreendendo sobre a medição, coleta e análise dos dados gerados.

Conjunto de Valores para Avaliação Analitica

O conjunto de valores utilizados para a avaliação estão descritos abaixo, irei utilizar o valor (n) número de cidades os restantes informarei de forma padronizada para compararmos o impacto (n) com diferentes entradas.

Valores a serem analisados

número de cidades: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100

Valores Padrões

número de formigas: 200
número de viagens: 300
alpha: 3 referente ao peso do feromônio
beta: 2 referente ao peso da distância percorrida
redução Feromônio: 0.01 referente a 10% de perda de feromônio
aumento Feromônio 2.0 referente 200% de ganho de feromônio

Métricas de Desempenho

Para as métricas de desempenho do tempo de processamento com a variação dos valores dos parâmetros com a utilização da meta-heuristica colônia de formigas em relação a otimização das rotas através do feromônio.

Implementação

Script para orquestrar a realização de avaliação analítica.

```
    trials = args.trials
    f = open(args.out, "w")
    f.write("#TSP_ACO\n")
    f.write("#n time_s_avg time_s_std (for {} trials)\n".format(trials))
    m = 100
    np.random.seed(args.seed)
    for n in range(args.nstart, args.nstop+1, args.nstep): #range(1, 100):
        resultados = [0 for i in range(trials)]
        tempos = [0 for i in range(trials)]
        for trial in range(trials):
            print("\n-------")
            print("n: {} trial: {}".format(n, trial+1))
            entrada = np.random.randint(0, n, n)
            print("Entrada: {}".format(entrada))
            tempo_inicio = timeit.default_timer()
            resultados[trial] = TSP_ACO(entrada)
            tempo_fim = timeit.default_timer()
            tempos[trial] = tempo_fim - tempo_inicio
            print("Saída: {}".format(resultados[trial]))
            print('Tempo: {} s'.format(tempos[trial]))
            print("")

        tempos_avg = np.average(tempos)  # calcula média
        tempos_std = np.std(a=tempos, ddof=False)  # ddof=calcula desvio padrao de uma amostra?


        f.write("{} {} {}\n".format(n, tempos_avg, tempos_std))
    f.close()

if __name__ == "__main__":
```

Script para gerar gráficos usando interpolação.

```
212  class TSP_ACO(Experimento):
213
214      def __init__(self, args):
215          super().__init__(args)
216          self.id = "a"
217          self.script = "tsp_aco.py"
218          self.output = "tsp_aco.txt"
219
220          indice_cor = 4
221
222          # configurações de plotagem
223          self.medicao_legenda = "TSP_ACO medido"
224          self.medicao_cor_rgb = mapa_escalar.to_rgba(3*indice_cor)
225          self.medicao_formato = formatos[indice_cor]
226
227          self.aproximacao_legenda = "TSP_ACO aproximado"
228          self.aproximacao_cor_rgb = mapa_escalar.to_rgba(2*indice_cor+1)
229
230          self.multiplo = 1
231          self.tamanhos_aproximados = range(self.args.nmax * self.multiplo+1)
232
233      def executa_aproximacao(self):
234          # realiza aproximação
235          parametros, pcov = opt.curve_fit(funcao_analitica, xdata=self.tamanhos, ydata=self.medias)
236          self.aproximados = [funcao_analitica(x, *parametros) for x in self.tamanhos_aproximados ]
237          print("aproximados:           {}".format(self.aproximados))
238          print("parametros_otimizados: {}".format(parametros))
239          print("pcov:                  {}".format(pcov))
240
241      def g(self, n, c):
242          return n*n*c
```

Requisitos necessários para reproduzir resultados.

para rodar os gráficos em python é necessário ter o python instalado no computador
e realizar alguns imports como

```
try:
    import sys
    import os
    import argparse
    import logging
    import subprocess
    import shlex
    import logging
    from abc import ABC, abstractmethod

    from scipy.special import factorial
    import math
    import numpy as np
    import matplotlib.pyplot as plt
    import scipy.optimize as opt
    import matplotlib.colors as colors
    import matplotlib.cm as cmx


except ImportError as error:
```

Exemplo de como executar o algoritmo em java
com o promt aberto no local do projeto basta executar
python tsp_aco_analitica.py
também podendo-se passar outros comandos juntamente com o citado acima.

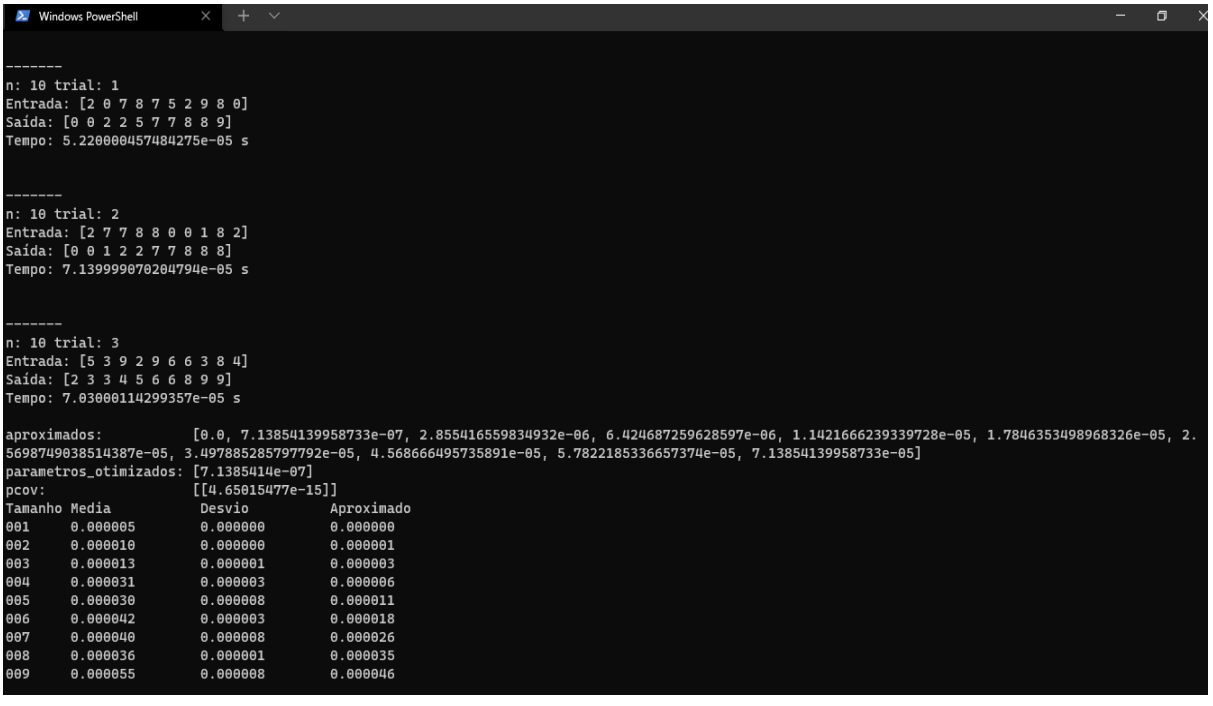
```
    comando_str = "python {}".format(self.script)
    comando_str += " --out {}".format(self.output)
    comando_str += " --nstart {}".format(self.args.nstart * self.multiplo)
    comando_str += " --nstop {}".format(self.args.nstop * self.multiplo)
    comando_str += " --nstep {}".format(self.args.nstep * self.multiplo)
    comando_str += " --trials {}".format(self.args.trials)
    if self.args.seed is not None:
        comando_str += " --seed {}".format(self.args.seed)
```

Exemplo de como gerar o gráfico em python
com o promt aberto no local do projeto basta executar
python tsp_aco_analitica.py – out “nomeGrafico”

Resultados da avaliação

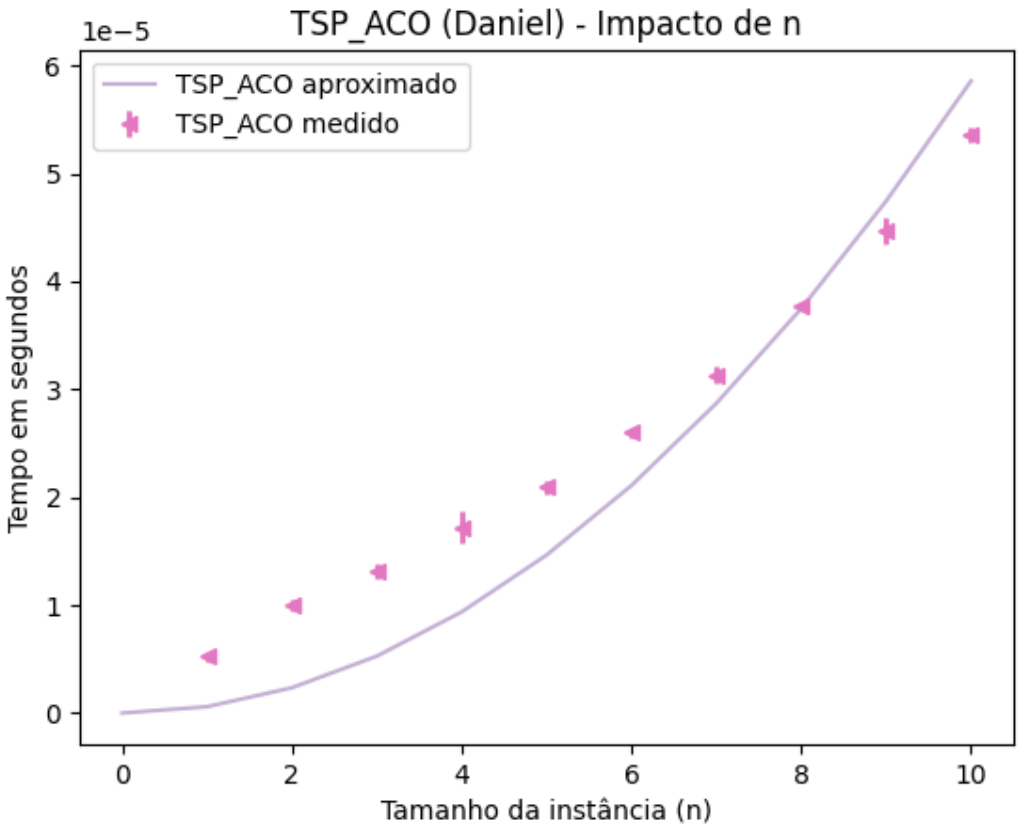
```
PS C:\Users\danie\OneDrive\Área de Trabalho\4ª Semestre\Analise e Projeto de Algoritmos\TSO_ACO_PYTHON> python .\tsp_aco_analitica.py
Argumentos:
       .\tsp_aco_analitica.py

Configurações:
       algoritmos: None
       nmax: 10
       nstart: 1
       nstep: 1
       nstop: 10
       out: None
       seed: None
       skip: False
       trials: 3
       verbosity: 20

Comando: python tsp_aco.py --out tsp_aco.txt --nstart 1 --nstop 10 --nstep 1 --trials 3
Solução final: 4 -> 8 -> 1 -> 5 -> 6 -> 7 -> 2 -> 3 | custo: 151
```

```
-------
n: 3 trial: 1
Entrada: [0 0 1]
Saída: [0 0 1]
Tempo: 1.4000004739500582e-05 s


-------
n: 3 trial: 2
Entrada: [1 0 2]
Saída: [0 1 2]
Tempo: 1.2699994840659201e-05 s


-------
n: 3 trial: 3
Entrada: [2 1 1]
Saída: [1 1 2]
Tempo: 1.2199991033412516e-05 s


-------
n: 4 trial: 1
Entrada: [3 0 1 2]
Saída: [0 1 2 3]
Tempo: 2.820001100189984e-05 s


-------
n: 4 trial: 2
Entrada: [2 0 3 1]
Saída: [0 1 2 3]
Tempo: 3.530000685714185e-05 s
```



Interpretação dos resultados

Substituição da figura contendo apenas os resultados avaliação experimental pela figura com resultados da avaliação experimental e analítica no documento do projeto.



Depois de ser tratado e interpretado, esse grande volume de dados o gráfico é capaz de mostrar o status do impacto de (n) no tempo de execução e os custos das rotas que as formigas são capazes de identificar o que pode ser melhorado. Além disso, essa mensuração de resultados ajuda na criação de simulações para que, posteriormente, gere melhores estratégias.

Referências

Artigo:
https://www.inf.ufpr.br/aurora/disciplinas/topicosia2/downloads/trabalhos/ACO_TSP.pdf

Livro:
Skiena2008_Book_TheAlgorithmDesignManual.
https://drive.google.com/drive/folders/1rfPYx59j67soYlmMNIeVsg--HpDrmOHF?usp=sharing
código:
https://github.com/glaucioscheibel/antcolony