Relatório EFC2

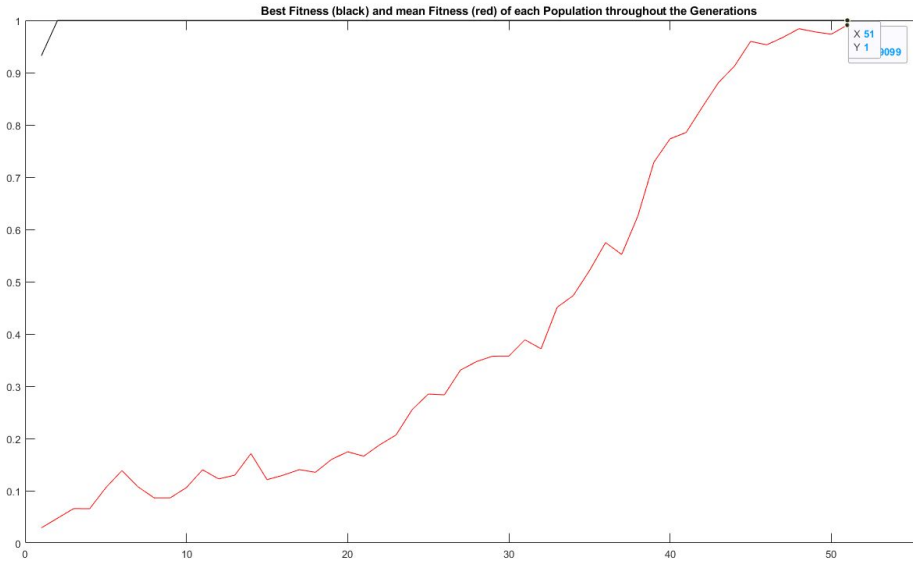
EA-072: Inteligência Artificial em Aplicações Industriais

André Barros de Medeiros RA:194060

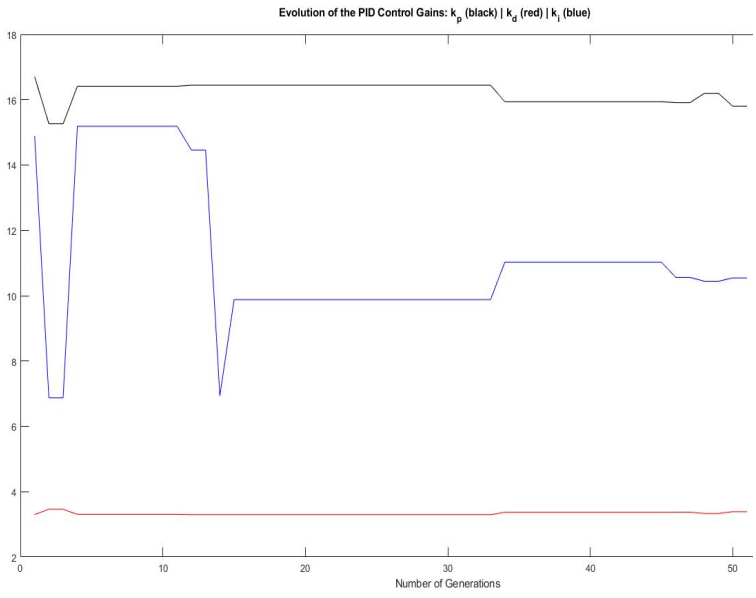
Q1. Controle PID empregando um algoritmo evolutivo, que opera como uma meta-heurística de otimização em espaços contínuos.

Gráficos Gerados:

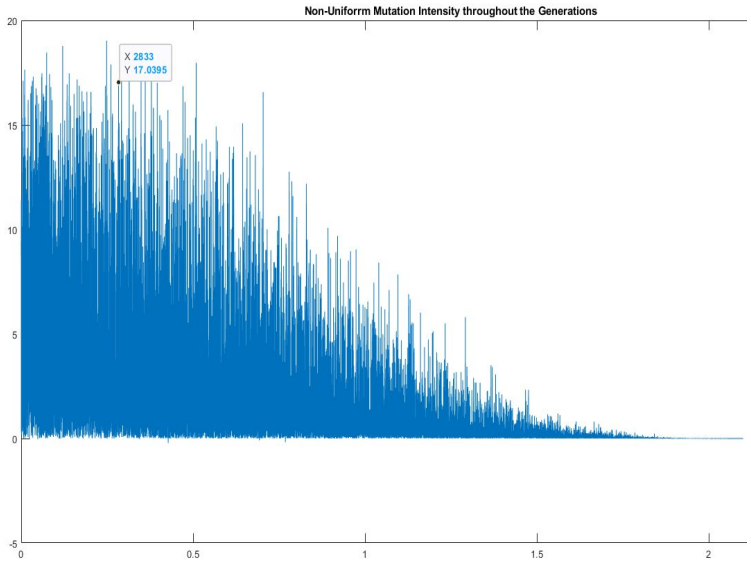
1. *Evolução do Melhor fitness e fitness médio*



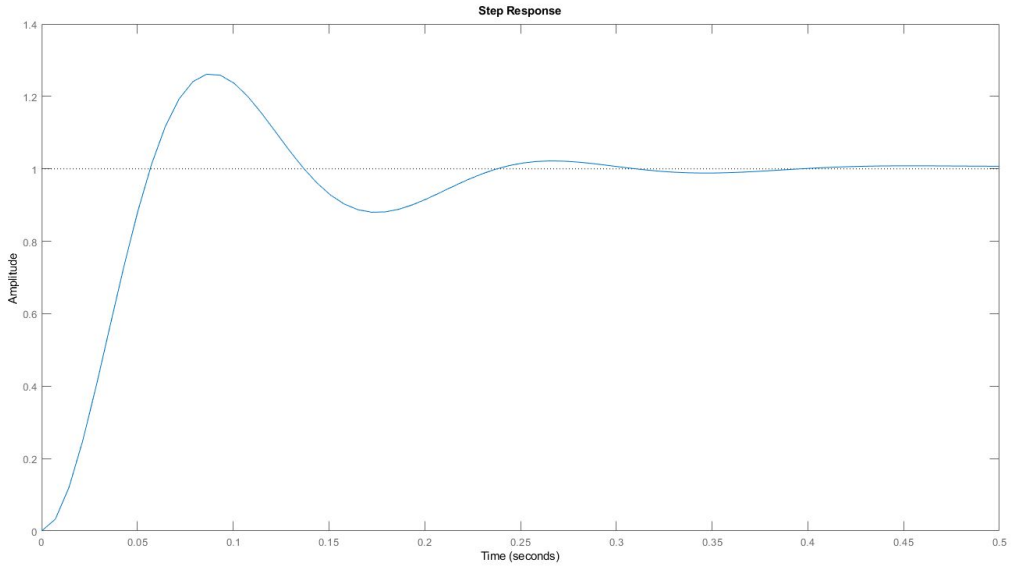
2. *Evolução dos Ganhos: Kp, Kd, Ki*



3. *Intensidade da mutação não uniforme ao longo das gerações:*



4. *Resposta a Degrau:*



<u>Análise dos Gráficos Gerados:</u>

- O primeiro mostra como o fitness máximo chegou em 1 muito rapidamente e que, ao final da execução, a última geração tem fitness médio muito próxima de 1 também.
- O segundo gráfico mostra a evolução dos valores dos 3 ganhos ao longo das gerações até chegar nos valores que ao final da execução fornecem um fitness ótimo. Temos que a evolução do fitness mostrado no primeiro gráfico surge a partir da evolução dos ganhos.
- O terceiro gráfico mostra o decaimento da intensidade de mutação ao longo das gerações. Isso ocorre pois, à medida que se aproxima dos ganhos ótimos, diminui-se a

necessidade/utilidade de uma taxa de mutação intensa já que o resultado que se procura está “por perto”.

- O quarto e último gráfico apresenta a resposta a degrau com a configuração final de ganhos. Percebe-se que a resposta é extremamente rápida (reparar na escala). Isso é condizente com o fitness 1 pois, se atingimos os parâmetros ótimos de um controle PID, estes devem fazer o controle do sistema de uma forma bastante ágil.

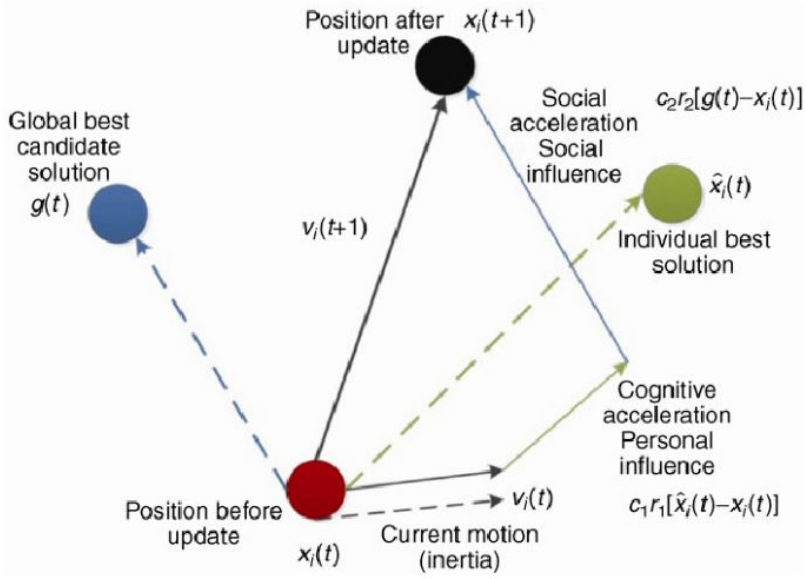
Q2. Controle PID empregando PSO (Particle Swarm Optimization), que opera como uma meta-heurística de otimização em espaços contínuos.

Segue a proposta de pseudocódigo para a implementação de Particle Swarm Optimization (Global Best):

```
FOR each particle i
    FOR each dimension d
      Initialize position Xid randomly within permissible range
      Initialize velocity Vid randomly within permissible range
    END
  END
  WHILE maximum iterations or minimum error criteria not reached
    iteration k
    FOR each particle i
      Calculate Fitness
      IF fitness is better than own best
        update own best (Pid)
      END
    END
    Choose the particle with the best fitness value (Pgd)
    FOR each Particle i
      FOR each dimension d
        Calculate velocity according to the equation: Vid(k+1)=
W*Vid(k)+C1*rand(Pid-Xid)+C1*rand(Pgd-Xid)
        Update particle position according to equation: Xid(k+1) = Xid(k)+Vid(k+1)
      END
    END
    Add to iteration
  END
```

A diferença do Global Best PSO para o Local Best PSO é na componente que influencia a nova velocidade da partícula. No primeiro o termo Pgd vai ser o mesmo para toda as partículas pois corresponde à posição da partícula com o melhor fitness de todas na dada iteração. Já no caso do segundo, teríamos um termo “Pld” que é único para cada partícula

(poderíamos ter um array PLD com o Pld de cada partícula), pois cada partícula se referencia apenas nas duas partículas vizinhas para atualizar a sua velocidade. Com isso para alterar o pseudocódigo acima para implementar o Local best, após atualizar o fitness de cada uma, deve ser calculado um Pld para cada partícula e isso deve ser armazenado para o uso no resto da iteração. Podemos reutilizar o mesmo array toda iteração, pois na próxima iteração o que interessa são os novos valores de fitness calculados, não precisando guardar os valores Pld da anterior. Abaixo temos uma ilustração para facilitar o entendimento do cálculo da nova velocidade:



Google images

Comparando o algoritmo PSO com o Algoritmo Genético da questão 1, percebemos que, tendo ajustado corretamente a questão da restrição do espaço de busca para o PSO, o primeiro tende a ser melhor do que o segundo. Podemos ver isso quando avaliamos a velocidade para atingir resultados de qualidade próxima da ótima (converge em um tempo menor). Além disso, com várias execuções, percebe-se que o PSO converge à solução ótima com uma frequência maior do que o Algoritmo da questão 1. Como observação, após pesquisa na internet, foi concluído que para datasets pequenos, a diferença é bem pequena, aumentando com o tamanho do dataset (quanto mais dados, melhor o PSO se torna, comparado ao Algoritmo Genético Padrão).

Também podemos falar do fato de que enquanto o PSO ter sido concebido para otimização de espaços contínuos (que é o caso dessa aplicação), o AG foi originalmente pensado para otimizar espaços discretos. Com isso, para a aplicação do segundo no nosso problema, teve de ser utilizado alguns operadores que foram propostos para que o AG funcionasse. Isso pode ser outro motivo pela melhor performance do PSO.

Entretanto, é importantíssimo que não falte a observação de que o PSO não necessariamente vai manter esta superioridade caso outros problemas de otimização sejam testados e casos operadores genéticos mais competentes sejam empregados. Além disso, temos que quando uma meta-heurística bate outras em algum cenário comparativo, pode ser

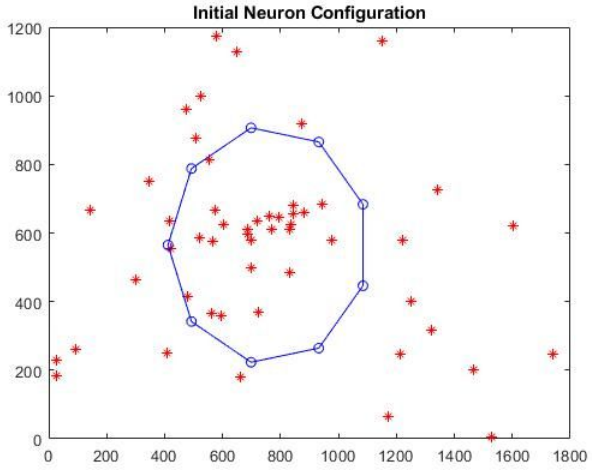
pelo fato dos operadores serem mais adequados para a aplicação pretendida, dadas suas peculiaridades, ou porque possui uma parametrização melhor definida que aquela adotada para o algoritmo concorrente. Ou seja, toda essa análise feita comparando o PSO com o AG é, inevitavelmente, subjetivo à nossa aplicação, podendo ser diferente caso a aplicação fosse outra.

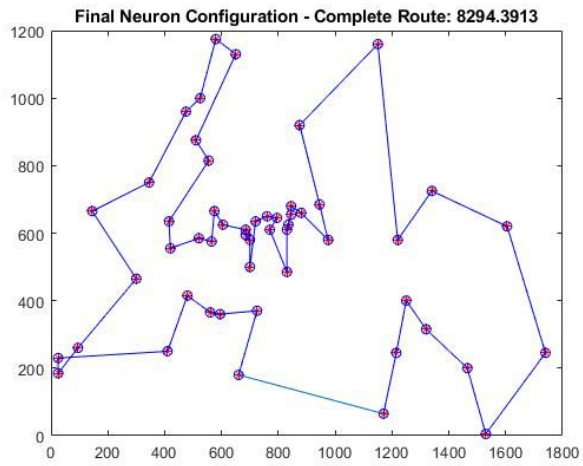
Finalmente, fazemos uma comparação entre o desempenho do Global Best e Local Best. A comunidade de pesquisa de PSO em geral recomenda o uso de Local Best pois tende a explorar um espaço maior, evita com mais frequência mínimos locais e, além disso, evita convergência prematura (o que ocorre bastante no uso de Global Best).

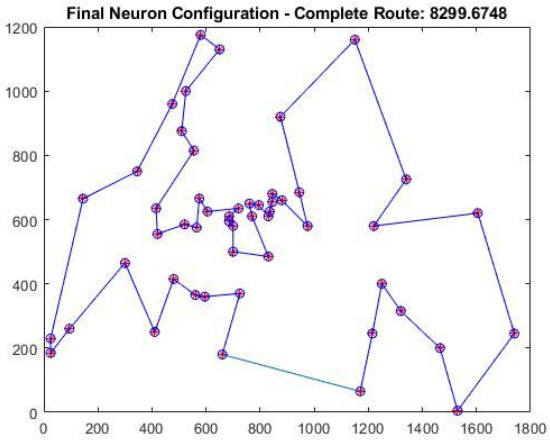
Q3. Otimização Combinatória

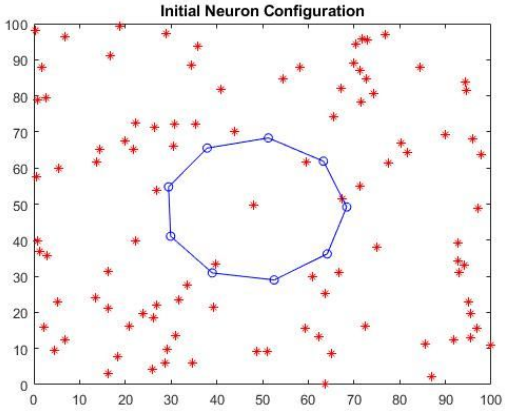
1. <u>Mapa de Kohonen</u>

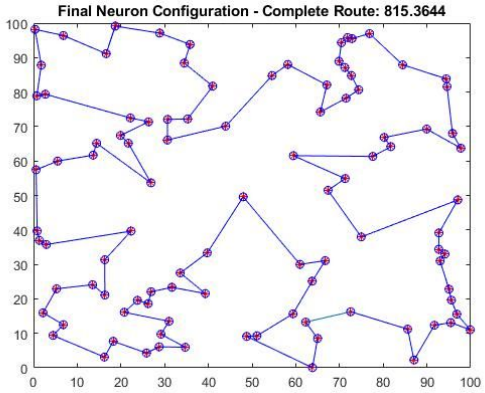
Seguem abaixo os resultados de múltiplas execuções do algoritmo do caixeiro viajante (duas execuções para cada mapa fornecido) na seguinte ordem: Berlin52, inst1, inst2.

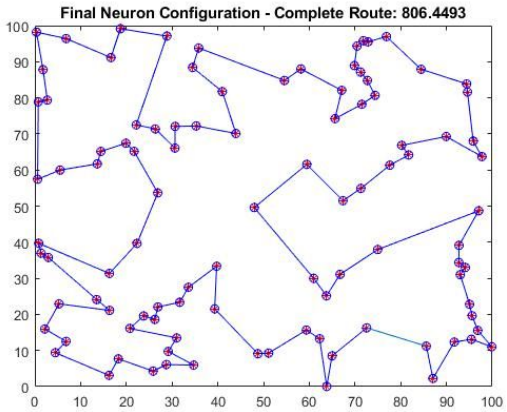


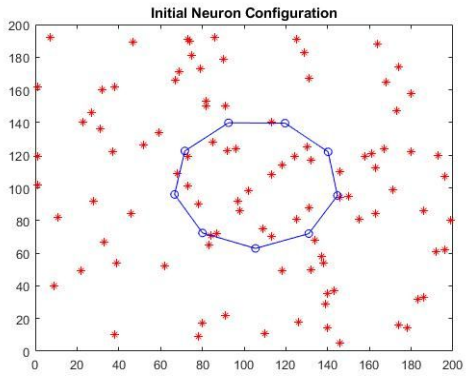


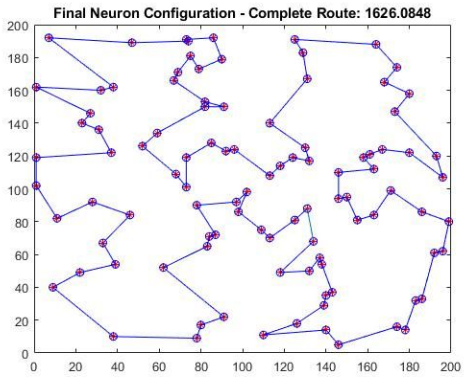


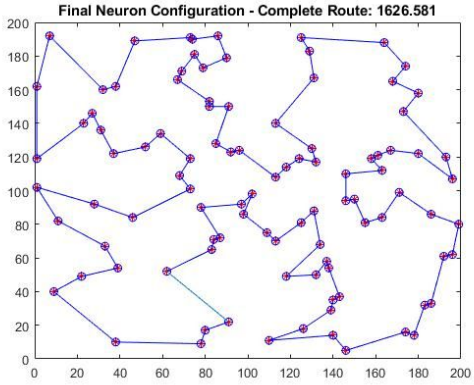

Initial Neuron Configuration


Final Neuron Configuration - Complete Route: 815.3644


Final Neuron Configuration - Complete Route: 806.4493


Initial Neuron Configuration


Final Neuron Configuration - Complete Route: 1626.0848


Final Neuron Configuration - Complete Route: 1626.581

2. Algoritmo Genético

Segue o pseudocódigo da resolução do problema do Caixeiro Viajante por Algoritmo Genético:

1. Criar população inicial de P cromossomos

2. Avaliar o fitness de cada cromossomo.

3. Escolher X pais da atual população por meio de seleção proporcional

4. Selecionar aleatoriamente dois pais para gerar filho, utilizando o operador de recombinação.

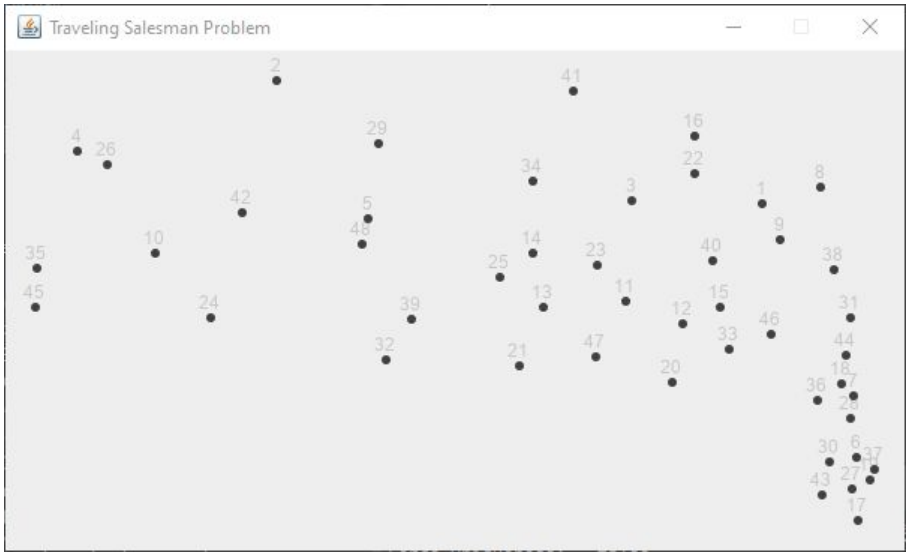
5. Aplicar o operador de mutação aos filhos

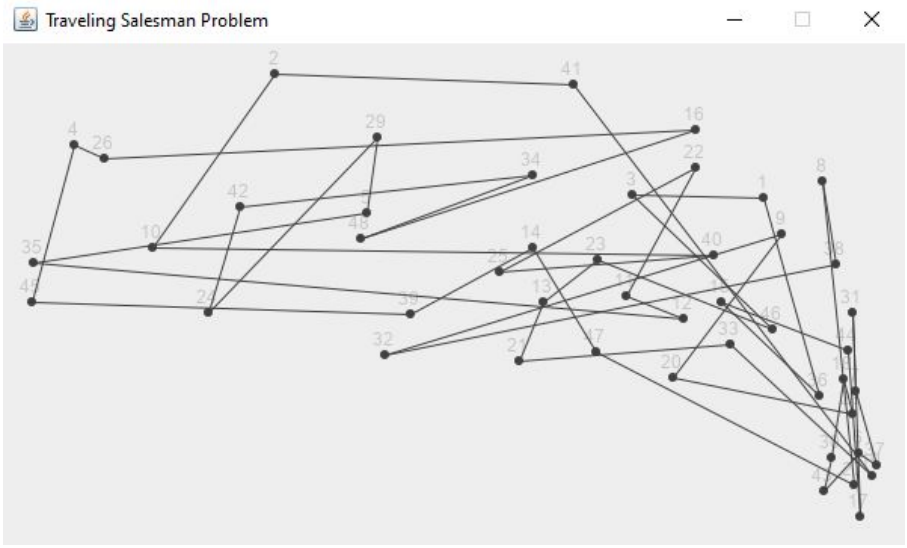
6. Repetir 4 e 5 até todos os pais selecionados terem gerado filhos

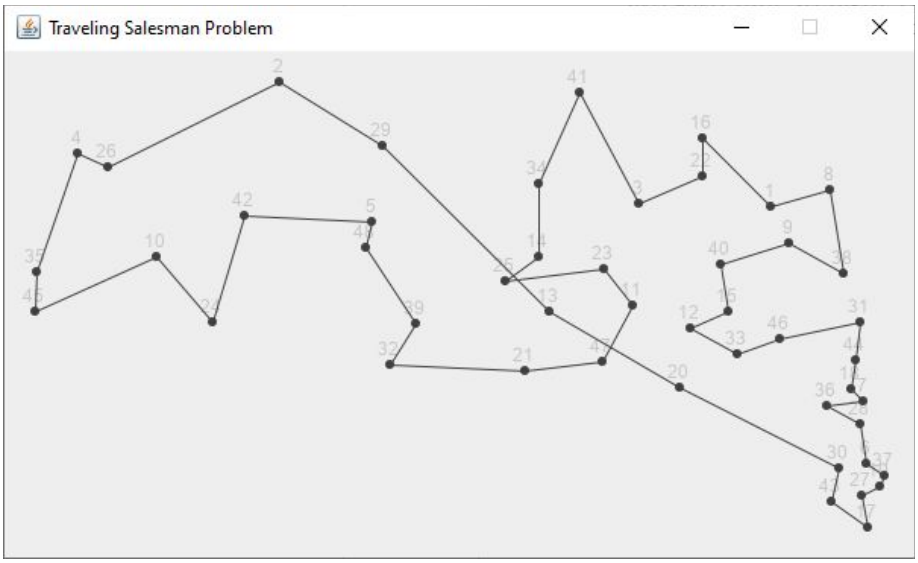
7. Trocar população anterior pela nova

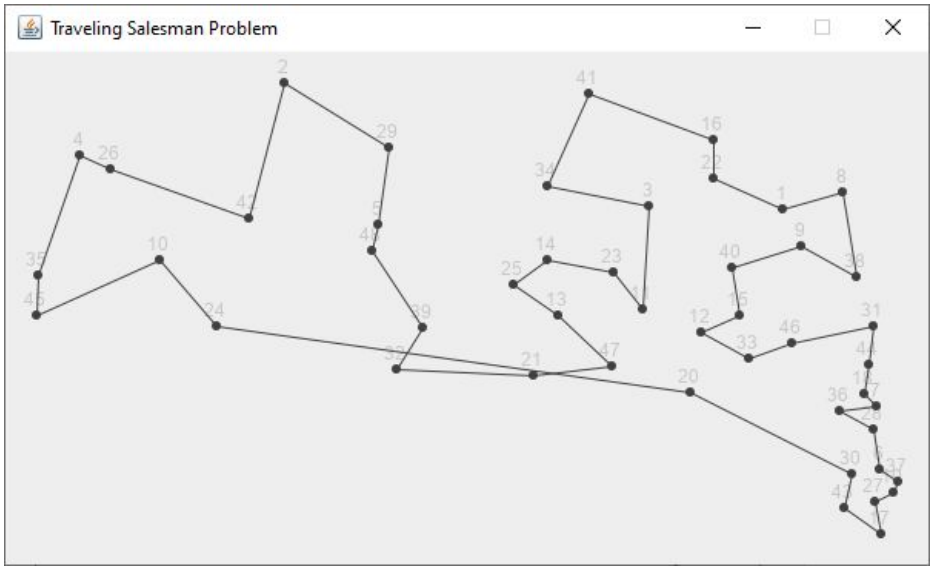
8. Voltar ao passo 2 IF não tiver alcançado critério de parada

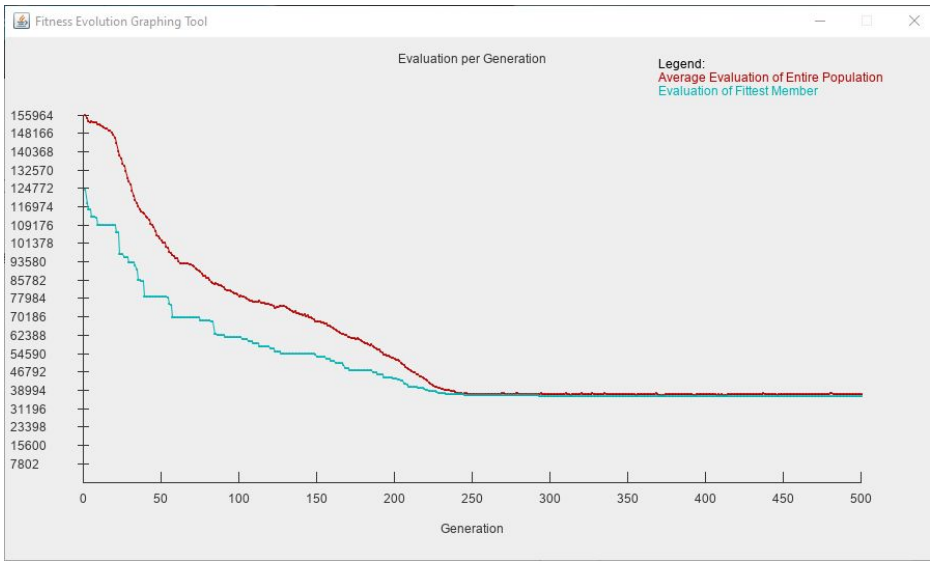
O programa para a implementação desse problema foi feito em java. Abaixo temos a configuração inicial, um resultado intermediário, dois resultados finais e o gráfico da evolução do fitness ao longo do tempo (nesta ordem). Observação: o gráfico de fitness é referente à primeira execução.




Traveling Salesman Problem


Traveling Salesman Problem


Traveling Salesman Problem



Seguem os códigos do operador de recombinação, operador de mutação e de busca local:

```
        /**
     * Mutate the Chromosome based on what type is selected.
     * @param chromosome    the Chromosome to mutate
     * @return              the mutated Chromosome
     */
    private Chromosome mutate (Chromosome chromosome) {
        if (mutationType == MutationType.INSERTION) {
            return Mutation.insertion(chromosome, random);
        } else if (mutationType == MutationType.RECIPROCAL_EXCHANGE) {
            return Mutation.reciprocalExchange(chromosome, random);
        } else if (mutationType == MutationType.SCRAMBLE) {
            return Mutation.scrambleMutation(chromosome, random);
        } else { // Default is insertion.
            return Mutation.insertion(chromosome, random);
        }
    }
```

```
/**
     * Perform the selected type of crossover.
     * @param p1            the first parent
     * @param p2            the second parent
     * @return              the children
     */
```

```
    private ArrayList<Chromosome> crossover (Chromosome p1, Chromosome p2)
{
        ArrayList<Chromosome> children;
        if (crossoverType == CrossoverType.UNIFORM_ORDER) {
            children = Crossover.uniformOrder(p1, p2, random);
        } else if (crossoverType == CrossoverType.ONE_POINT) {
            children = Crossover.onePointCrossover(p1, p2, random);
        } else {
            children = Crossover.orderCrossover(p1, p2, random);
        }
        return children;
    }
```

```
private Chromosome performLocalSearch (Chromosome chromosome) {

        int bestDistance = chromosome.getDistance();
        City[] array = chromosome.getArray();
        City[] bestArray = array.clone();

        for (int i = 0; i < array.length-1; i++) {
            for (int k = i+1; k < array.length; k++) {

                City[] temp = array.clone();

                // Reverse order from i to k.
                for (int j = i; j <= (i+k)/2; j++) {
                    swap(temp, j, k - (j-i));
                }

                Chromosome c = new Chromosome(temp);

                int distance = c.getDistance();
                if (distance < bestDistance) {
                    bestDistance = distance;
                    bestArray = c.getArray();
                }

            }
        }
        return new Chromosome(bestArray);}
```

Q4. Regressão Simbólica utilizando *Eureqa*

1. Mapeamento $\Re^1 \rightarrow \Re^1$ [y = f(t)] ; y = t*cos(5.3+6t) + noise

Solution	y = 0.696981168549497*t*cos(2256.0713034038*t) + 0.322513018100629*t*cos(2224.65875947568*t)
R^2 Goodness of Fit	0.92451851
Correlation Coefficient	0.96363886
Maximum Error	2.0992338
Mean Squared Error	0.81168394
Mean Absolute Error	0.71182053
Coefficients	4
Complexity	21
Primary Objective	0.71182053
Fit (Normalized Primary Obj.)	0.28637649

2. Mapeamento $\Re^3 \rightarrow \Re^1$ [y=f(x1,x2,x3)] ; y = x1+cos(x2)+sin(x3)+noise

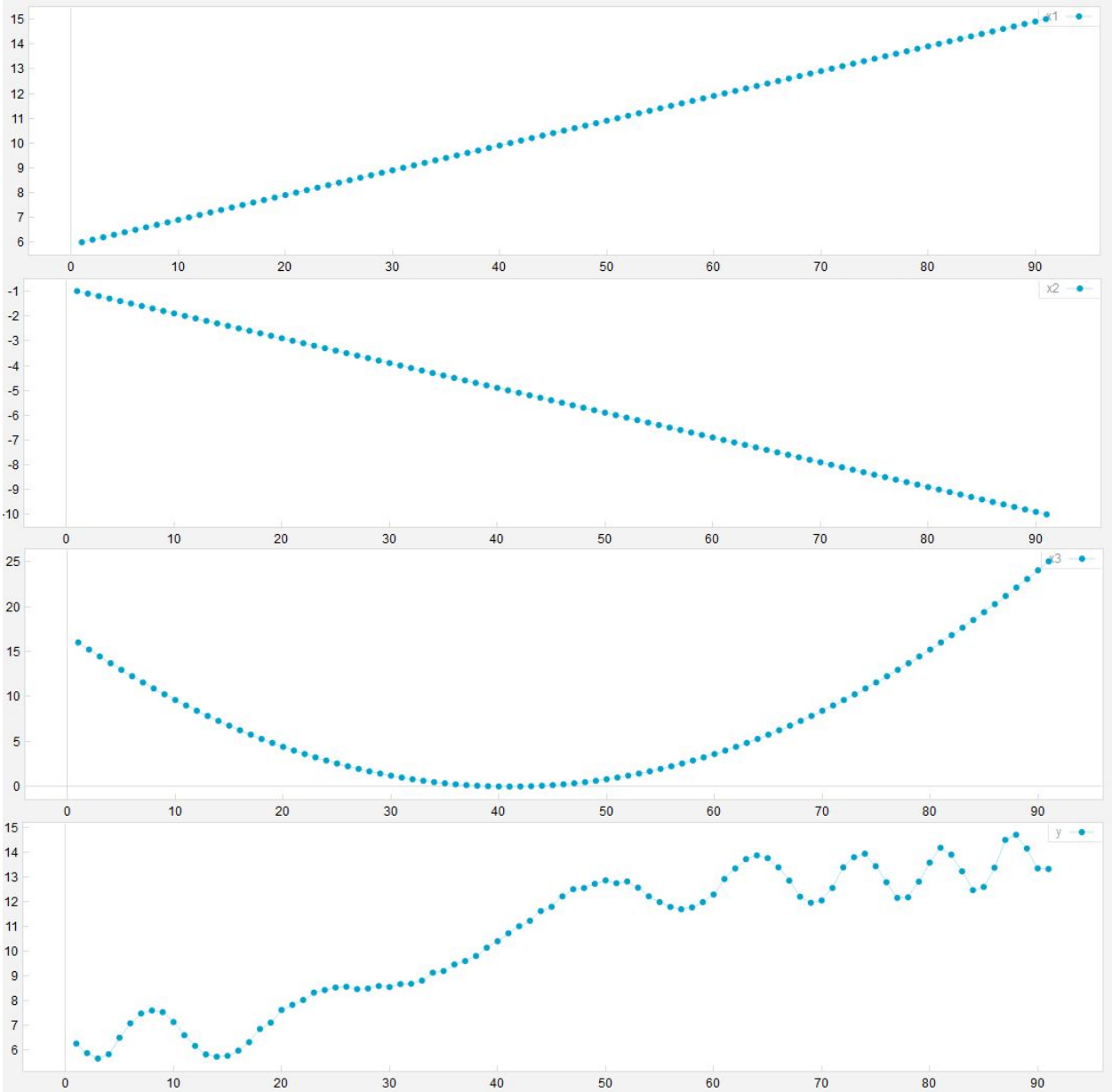
   Geramos os dados de X1, X2, X3 e Y com a seguinte função do Matlab:

```
i=0;
for j=1:0.1:10
    i=i+1;
    noise=0.2*rand(91,1);
    x1(1,i)=j+5;
    x2(1,i)=-j;
    x3(1,i)=(j-4.5)^2;
    y(1,i) = x1(1,i)+cos(x2(1,i))+sin(x3(1,i))+noise(i,1);
    x1=transpose(x1);
    x2=transpose(x2);
    x3=transpose(x3);
```

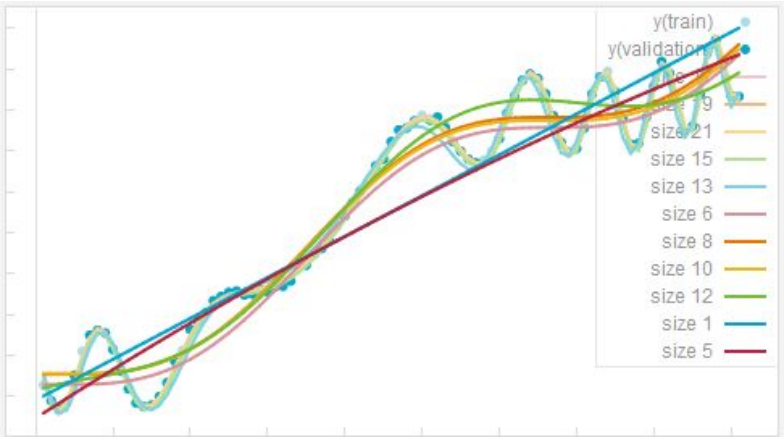
```
    y=transpose(y);
end
```

Observação: deve-se executar o código 2 vezes para preencher as variáveis por completo (devido ao funcionamento da função transpose()).

Com isso, passamos os dados para o Eureqa. A seguir está a representação gráfica de X1, X2, X3 e Y (nessa ordem):



Com isso, foi feita a inferência com o software Eureqa. Abaixo seguem algumas soluções que foram apresentadas pelo software:



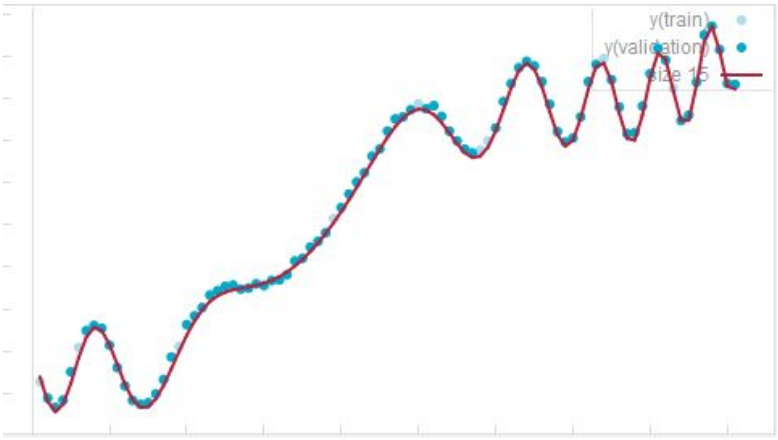
size	Fit	Solutions
17	0.078	y = 0.0933162114575471 + x1 + cos(x2) - sin(x3 + 0.971169729420272*x1)
19	0.077	y = 0.0882137532136215 + x1 + cos(x2) - 0.990256927794345*sin(x3 + 0.97176300019404*x1)
21	0.076	y = 0.0633809159923694 + 1.00161530792091*x1 + cos(x2) - 0.979156757524207*sin(x3 + 0.971756078055139*x1)
15	0.123	y = x1 + cos(x2) - sin(x3 + 0.973396505573773*x1)
13	0.26	y = x1 + cos(x2) - sin(x1 + x3)
6	0.761	y = x1 - sin(x1)
8	0.727	y = 0.250502437147123 + x1 - sin(x1)
10	0.727	y = 0.366458210023106 + 0.984591171636002*x1 - sin(x1)
12	0.705	y = 1.05151604528235*x1 + cos(x2) - 0.041051585464885*x3

Levando em consideração simplicidade e acurácia escolhemos uma solução com boa relação custo-benefício entre esses dois:

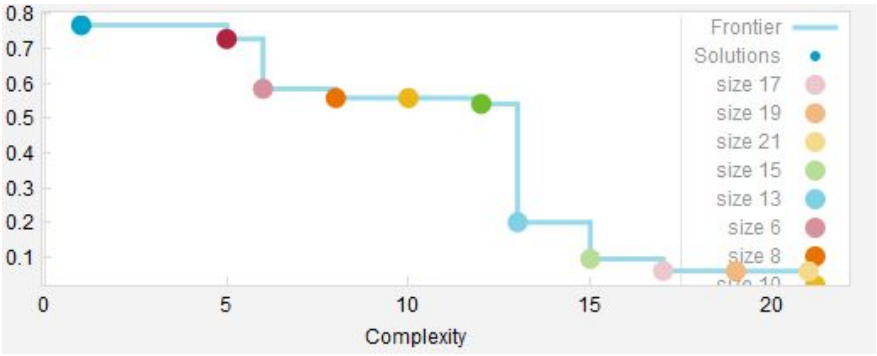
Solution	y = x1 + cos(x2) - sin(x3 + 0.973396505573773*x1)
R^2 Goodness of Fit	0.99830041
Correlation Coefficient	0.9996219
Maximum Error	0.28166569
Mean Squared Error	0.012776339
Mean Absolute Error	0.094251929
Coefficients	1

Complexity	15
Primary Objective	0.094251929
Fit (Normalized Primary Obj.)	0.12284597

Gráficamente, ela se apresenta da seguinte maneira:



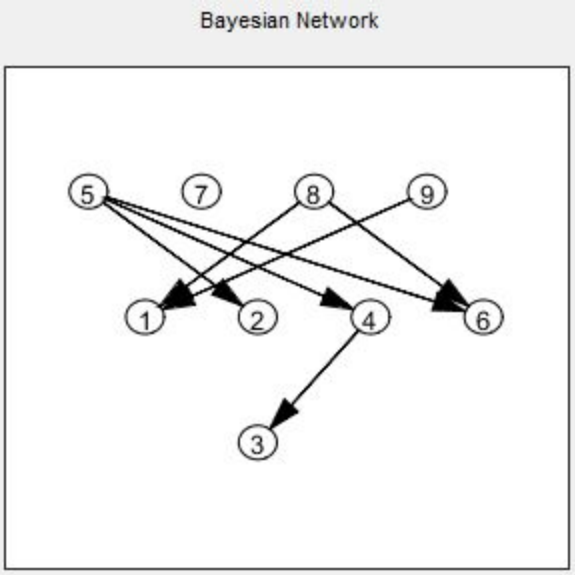
Além disso, apresentamos a fronteira de Pareto obtida. Lembramos que a Fronteira de Pareto é o conjunto de parametrizações que são pareto-eficientes.



Q5. Redes Bayesianas

Temos que os dados contidos no arquivo dados_BN_EFC2.mat possuem 10000 amostras, 9 atributos, onde todos exceto o atributo 2 são atributos binários. Já o próprio atributo 2 assume 7 estados diferentes (veja tabela de probabilidade “Variable 7”).

Abaixo segue a rede bayesiana gerada a partir dos dados citados, junto com as tabelas de probabilidades para cada nó:



Variable 1		FALSE	TRUE
V8 false	V9 false	1	0
V8 false	V9 true	0	1
V8 true	V9 false	0	1
V8 true	V9 true	1	0

Variable 2	State1	State2	State3	State4	State5	State6	State7
V5 false	0.2633	0.1904	0.1769	0.1614	0.1182	0.0655	0.0216
V5 true	0.1159	0.1414	0.1283	0.1261	0.1657	0.1548	0.1677

Variable 3	FALSE	TRUE
V4 false	0.3373	0.6627
V4 true	0.9802	0.0198

Variable 4	FALSE	TRUE
V5 false	0.5949	0.4051
V5 true	0.7509	0.2491

Variable 5	FALSE	TRUE
	0.1481	0.8519

Variable 6		FALSE	TRUE
V5 false	V8 false	0.8788	0.1212
V5 false	V8 true	0.5888	0.4112
V5 true	V8 false	0.0528	0.9472
V5 true	V8 true	0.1587	0.8413

Variable 7	FALSE	TRUE
	0.4972	0.5028

Variable 8	FALSE	TRUE
	0.482	0.518

Variable 9	FALSE	TRUE
	0.5022	0.4978

Ao analisar a rede bayesiana gerada, percebe-se que o nó 7 não possui conexões a nenhum outro nó. Isto se dá pelo fato de foi inferido, no processo de construção da rede, que a variável 7 não influencia na probabilidade de ocorrência de nenhuma outra.

Além disso temos que a probabilidade da variável 5 ser verdade é de 85.19%. É importante ressaltar que esta probabilidade não é influenciada por nenhuma outra variável (veja rede).

Temos também que, ao analisar a tabela de probabilidade do atributo 6, a probabilidade do atributo 6 ser verdade dado que o atributo 5 é falso e o atributo 8 é verdade é de 41.12%.

Não podemos deixar de observar também que, ao analisar a tabela de probabilidade da variável 1, podemos que há uma relação de XOR entre as variáveis 8 e 9.

Finalmente, também calculamos a probabilidade do atributo 5 ser verdade, dado que 3 é verdade. Para isso, partimos do Teorema de Bayes, e seguimos um processo similar ao apresentado no Exercício 2 do Tópico 10 da página do curso. Concluímos finalmente que P(5=1|3=1) = 87.38%.

Q6. Árvores de Decisão

Para os dados contidos no arquivo dados_AD_EFC2.arff, há 7 atributos, sendo eles 5 atributos numéricos (temperatura média, umidade média, altura da chuva mensal, precipitação em 21 dias e número de dias de chuva) e 2 atributos categóricos (local e classe). Abaixo seguem as estatísticas referentes a esses atributos:

Name: temperatura_media Type: Numeric
Missing: 0 (0%) Distinct: 14 Unique: 0 (0%)

Statistic	Value
Minimum	12.82
Maximum	21.98
Mean	18.448
StdDev	2.965

Name: umidade_media Type: Numeric
Missing: 0 (0%) Distinct: 14 Unique: 0 (0%)

Statistic	Value
Minimum	68.38
Maximum	97.83
Mean	90.306
StdDev	7.247

Name: altura_chuva_mensal Type: Numeric
Missing: 0 (0%) Distinct: 14 Unique: 0 (0%)

Statistic	Value
Minimum	9
Maximum	273.7
Mean	107.779
StdDev	71.769

Name: precip_21_dias Type: Numeric
Missing: 0 (0%) Distinct: 14 Unique: 0 (0%)

Statistic	Value
Minimum	0
Maximum	216.9
Mean	85.179
StdDev	62.745

Name: numero_dias_chuva Type: Numeric
Missing: 0 (0%) Distinct: 11 Unique: 0 (0%)

Statistic	Value
Minimum	3
Maximum	15
Mean	8.857
StdDev	3.823

Name: local
Missing: 0 (0%) Distinct: 5 Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	ID	14	14.0
2	IM	14	14.0
3	PM	14	14.0
4	CHI	14	14.0
5	GAL	14	14.0

Name: class
Missing: 0 (0%) Distinct: 2 Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	0	11	11.0
2	1	59	59.0

Ao rodarmos a árvore de decisão C4.5 com os parâmetros indicados no enunciado, obtivemos o seguinte resultado:

- % de classificação correta: 91.4286 %
- Matriz de confusão: a b <-- classified as
  8 3 | a = 0
  3 56 | b = 1
- A própria árvore de decisão:

J48 unpruned tree
---
temperatura_media <= 21.87
| numero_dias_chuva <= 4
| | local = ID: 1 (3.0)
| | local = IM: 1 (3.0)
| | local = PM
| | | numero_dias_chuva <= 3: 0 (1.0)
| | | numero_dias_chuva > 3: 1 (2.0)
| | local = CHI
| | | numero_dias_chuva <= 3: 1 (1.0)
| | | numero_dias_chuva > 3: 0 (2.0)
| | local = GAL: 0 (3.0)
| numero_dias_chuva > 4
| | local = ID: 1 (10.0)
| | local = IM
| | | umidade_media <= 97.5: 1 (9.0)
| | | umidade_media > 97.5: 0 (1.0)
| | local = PM: 1 (10.0)
| | local = CHI: 1 (10.0)

| | local = GAL: 1 (10.0)
temperatura_media > 21.87
| local = ID: 1 (1.0)
| local = IM: 0 (1.0)
| local = PM: 0 (1.0)
| local = CHI: 0 (1.0)
| local = GAL: 0 (1.0)

Number of Leaves : 18
Size of the tree : 26
Time taken to build model: 0.01 seconds

É interessante ressaltar que a matriz de confusão (apresentada acima) indica que em 8 vezes, foi classificado que não havia a presença da espécie (a=0) corretamente e 3 vezes erroneamente. Além disso em 56 vezes foi classificado que havia a presença da espécie (b=1) corretamente e 3 vezes erroneamente.

Para facilitar mais ainda a compreensão, estruturamos a árvore de decisão acima na forma de regras de antecedente e consequente: (onde 0 indica classe a e 1 indica classe b)

SE <temperatura_media<21.87 & numero_dias_chuva<=4> ENTÃO < ID=1 & IM=1 & GAL=0>
SE <temperatura_media<21.87 & numero_dias_chuva <=4 & numero_dias_chuva>3> ENTÃO <CHI=0>
SE <temperatura_media<21.87 & numero_dias_chuva<=4 & numero_dias_chuva<=3> ENTÃO <CHI=1>
SE <temperatura_media<21.87 & numero_dias_chuva>4> ENTÃO <ID=1 & PM=1 & CHI=1 & GAL=1>
SE <temperatura_media<21.87 & numero_dias_chuva>4 & umidade_media<=97.5> ENTÃO <IM=1>
SE <temperatura_media<21.87 & numero_dias_chuva>4 & umidade_media>97.5> ENTÃO <IM=0>
SE <temperatura_media>21.87> ENTÃO <ID=1 & IM=0 & PM=0 & CHI=0 & GAL=0>

Em seguida, alteramos o parâmetro minNumObj e unpruned (um de cada vez) e obtivemos os resultados abaixo:

unpruned=false (pruned):
J48 pruned tree
---
temperatura_media <= 21.87: 1 (65.0/7.0)
temperatura_media > 21.87: 0 (5.0/1.0)

```
Correctly Classified Instances         61               87.1429 %

=== Confusion Matrix ===
a  b   <-- classified as
4  7  |  a = 0
2 57 |  b = 1
```

minNumObj = 2:

```
J48 unpruned tree
------------------
temperatura_media <= 21.87
|   numero_dias_chuva <= 4
|   |   local = ID: 1 (3.0)
|   |   local = IM: 1 (3.0)
|   |   local = PM: 1 (3.0/1.0)
|   |   local = CHI: 0 (3.0/1.0)
|   |   local = GAL: 0 (3.0)
|   numero_dias_chuva > 4: 1 (50.0/1.0)
temperatura_media > 21.87: 0 (5.0/1.0)

Correctly Classified Instances         63               90      %

=== Confusion Matrix ===
a  b   <-- classified as
8  3  |  a = 0
4 55 |  b = 1
```

Percebe-se que no caso em que aumentamos o MinNumObj de 1 para dois, a árvore é simplificada, porém cai um pouco de performance. Entretanto, como a queda é pouca, essa configuração pode ser interessante para ser aplicada. Enquanto isso, quando ativamos o “pruning”, a árvore é muito simplificada, e o desempenho cai em torno de 4 vezes mais. Sendo assim, podemos dizer que nesse caso, dependendo não é muito interessante essa configuração a não ser que precisamos de algo muito simplificado (nesse caso, uma performance de 87% é de fato excelente). Abaixo, explicamos o que quer dizer cada um dos parâmetros:

Quando a árvore é “unpruned” ela tende a ser maior, pois ao final da sua construção, ela não passa por um processo de “poda”. Caso selecionarmos a opção “pruned”, ao final da construção da árvore adicionamos um passo no algoritmo que analisa se há partes dela que podem ser removidas sem afetar em muito a performance. A ideia por trás de pruning é, além de deixar a árvore mais compreensível, diminuir o risco de overfitting.

Já o parâmetro MinNumObj, de modo simplificado, regula o número mínimo de instâncias em uma folha. Entretanto, na verdade isso só ocorre para no mínimo dois ramos

(mas nada garante mais de dois). Com isso, o parâmetro MinNumObj pode ser melhor interpretado como a separação mínima de dados por ramo, especialmente para árvores onde nós têm vários filhos (árvores não binárias, conhecidas como “multi-way trees”).

Com isso, tomamos outro conjunto de dados principalmente de atributos categóricos. No caso, a base de dados é sobre a eleição presidencial de 1984 no Estados Unidos. Este arquivo pode ser encontrado no seguinte link: https://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html ou na pasta em que foi enviado esse relatório.

Quando construímos a árvore com as configurações iniciais do exercício anterior, conseguimos 95.1724% de acerto. Ao alterar os parâmetros (unpruned=false e minNumObj=3), conseguimos subir a taxa de acerto para 96.3218%. Lembrando que só havia 5% de “espaço” para melhorar, um aumento de 1.1494% já está muito bom. Com essa nova configuração, obtivemos a seguinte árvore de decisão:

```
J48 pruned tree
------------------
physician-fee-freeze = n: democrat (253.41/3.75)
physician-fee-freeze = y
|   synfuels-corporation-cutback = n: republican (145.71/4.0)
|   synfuels-corporation-cutback = y
|   |   mx-missile = n
|   |   |   adoption-of-the-budget-resolution = n: republican (22.61/3.32)
|   |   |   adoption-of-the-budget-resolution = y: democrat (7.25/2.23)
|   |   mx-missile = y: democrat (6.03/1.03)
```

Com a seguinte matriz de confusão:

```
=== Confusion Matrix ===
a     b   <-- classified as
259   8 |   a = democrat
8   160 |   b = republican
```