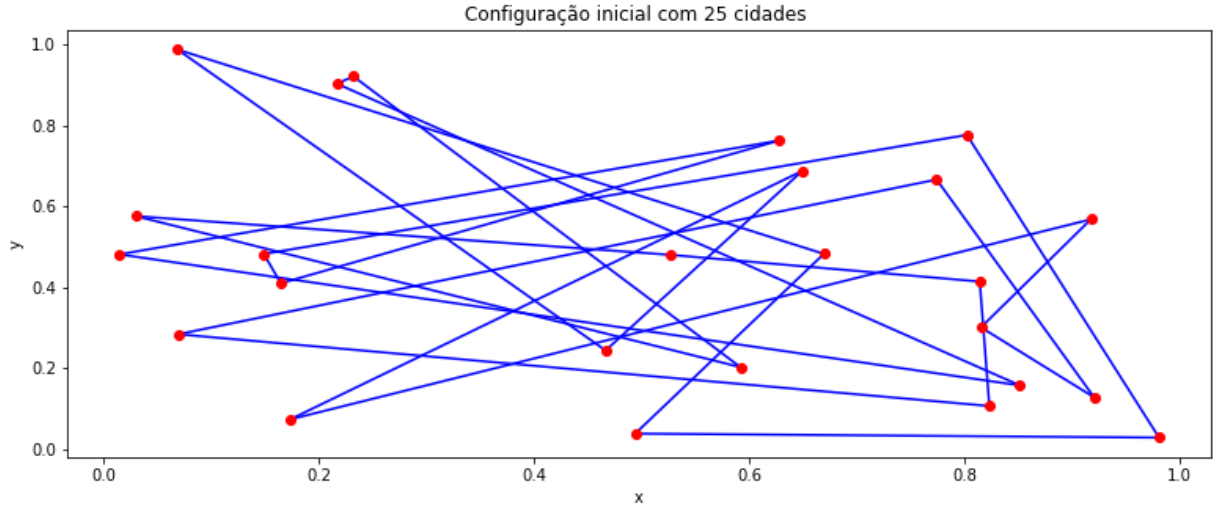
In [91]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [94]:
```
"""
@staticmethod : It can be called either on the class (e.g. C.f()) or on an instance
(e.g. C().f()).
"""
class Coordenada:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    @staticmethod  #
    def get_dist(a, b):
        return np.sqrt((a.x-b.x)**2 + (a.y-b.y)**2)

    @staticmethod
    def get_total_dist(coords):
        ener = 0  # distancia
        for first, second in zip(coords[:-1], coords[1:]):
            ener += Coordenada.get_dist(first, second)
        ener += Coordenada.get_dist(coords[0], coords[-1])
        return ener
```

In [125…
```
# numero de cidades (N)
N = 25

# preencher as coordenadas das cidades
coords = []
for _ in range(N):
    coords.append(Coordenada(np.random.uniform(), np.random.uniform()))
```

In [126…
```
# Plot caminho
fig = plt.figure(figsize=(60, 5))
ax1 = fig.add_subplot(1,4,1)
plt.xlabel('x')
plt.ylabel('y')
plt.title(f'Configuração inicial com {N} cidades ')
# configuração inicial
for first, second in zip(coords[:-1], coords[1:]):
    ax1.plot([first.x, second.x], [first.y, second.y], 'b')
ax1.plot([coords[0].x, coords[-1].x], [coords[0].y, coords[-1].y], 'b')

for c in coords:
    ax1.plot(c.x, c.y, 'ro')
plt.show()
```



In [127…

```
# vetor onde será armazenado o progresso das distancias percorridas pelo caixeiro
dists = []
```

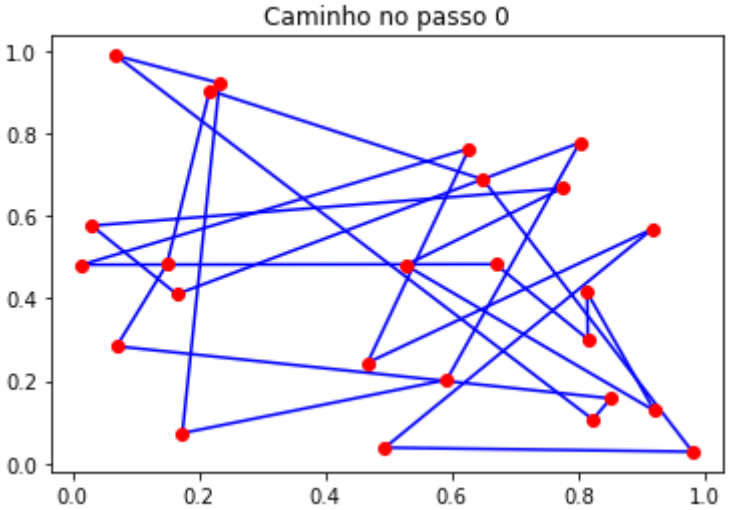
Simulated Annealing

In [128…

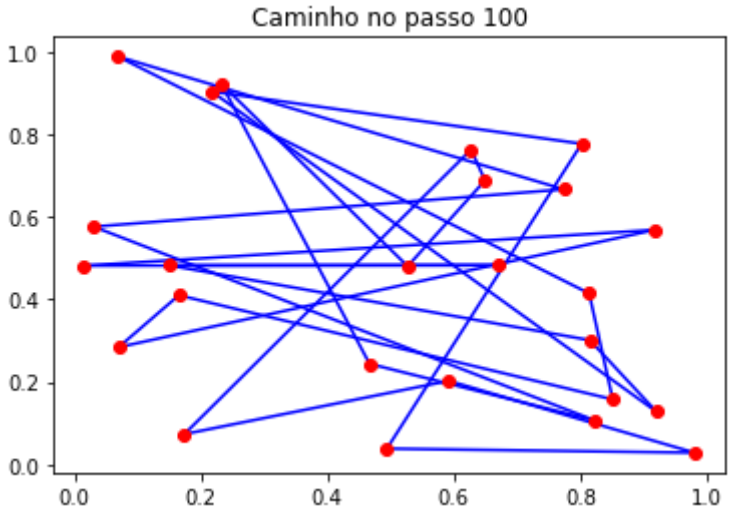
```
"""
O loop global fica setado em um valor arbitrariamente grande, de acordo com o protoc
de redução de temperatura, mas independentemente, ao atingir o valor limite, a itera
"""
ax2 = fig.add_subplot(1,2,2)
# plt.xlabel('x')
# plt.ylabel('y')
# plt.title('Caminho otimizado')

custo0 = Coordenada.get_total_dist(coords)

T = 10  # setar a temperatura inicial, entre [1,10]
dt = 0.99  # protocolo de reduçaõ de temperatura, entre [0.8 , 0.99]
Tf = 0.001
for i in range(10_000):
    T = T * dt

    if T <= Tf:
      print(f"\033[31mtemperatura Limite na iteração {i}")
      break

    # M.C
    for j in range(100):
        # troca duas coordenadas a pega uma nova solução de vizinho
        r1, r2 = np.random.randint(0, len(coords), size=2)
        temp = coords[r1]
        coords[r1] = coords[r2]
        coords[r2] = temp

        # novo custo
        custo1 = Coordenada.get_total_dist(coords)

        if custo1 < custo0:
            custo0 = custo1
        else:
            # aceita a nova config (pior) com uma dada probabilidade
            x = np.random.uniform()
            if x < np.exp((custo0 - custo1) / T):
                custo0 = custo1
            else:
```

```
                temp = coords[r1]
                coords[r1] = coords[r2]
                coords[r2] = temp
    dists.append(custo0)
    if i % 100 == 0:
        print(f"iteração: {i} \t energia/distância = {custo0:2f} ")
        #print(i, "custo =", custo0)

        # Plot the result
        plt.title(f'Caminho no passo {i}')
        for first, second in zip(coords[:-1], coords[1:]):
            plt.plot([first.x, second.x], [first.y, second.y], 'b')
        plt.plot([coords[0].x, coords[-1].x], [coords[0].y, coords[-1].y], 'b')

        # printa os pontos/cidades
        for c in coords:
            plt.plot(c.x, c.y, 'ro')
        plt.show()

# Plot the result

# plt.xlabel('x')
# plt.ylabel('y')
plt.title('Caminho otimizado')
for first, second in zip(coords[:-1], coords[1:]):
    plt.plot([first.x, second.x], [first.y, second.y], 'b')
plt.plot([coords[0].x, coords[-1].x], [coords[0].y, coords[-1].y], 'b')

# printa os pontos/cidades
for c in coords:
    plt.plot(c.x, c.y, 'ro')
plt.show()
```

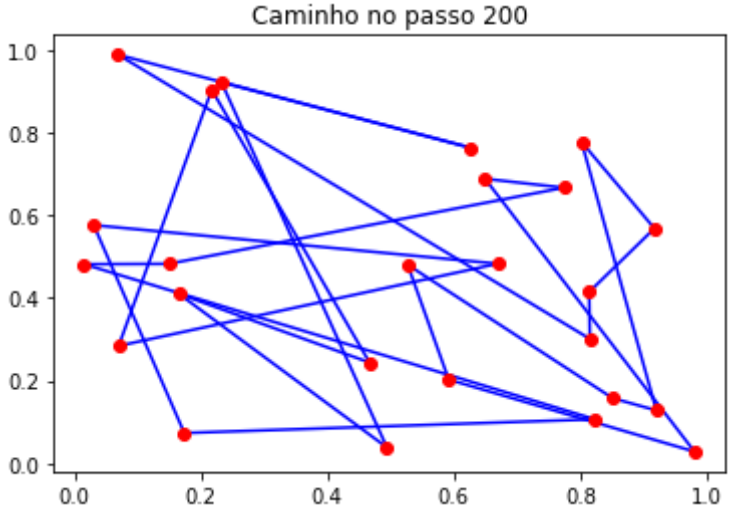
```
iteração: 0     energia/distância = 12.730322
```



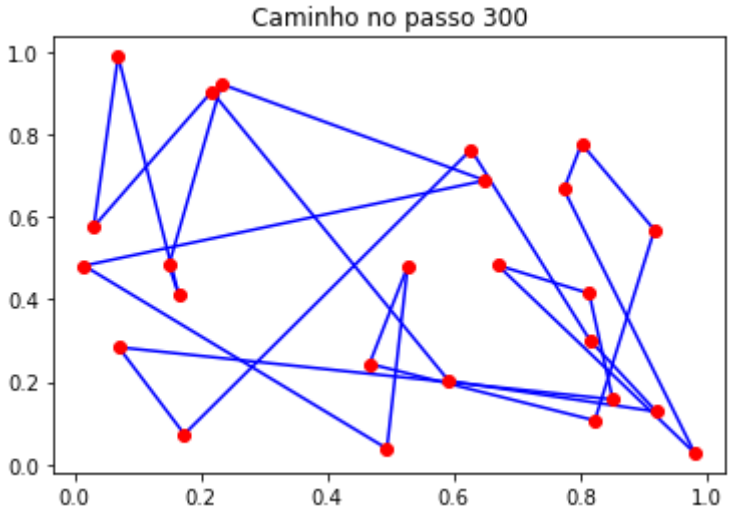

```
iteração: 100    energia/distância = 14.977912
```



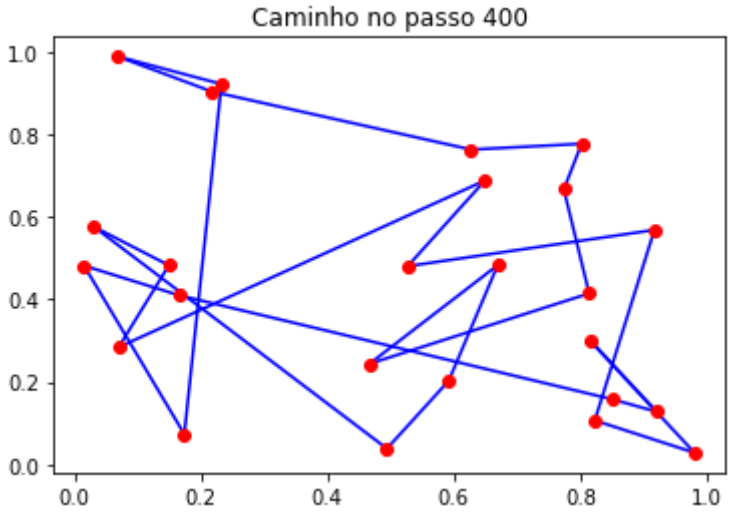

```
iteração: 200     energia/distância = 12.581400
```



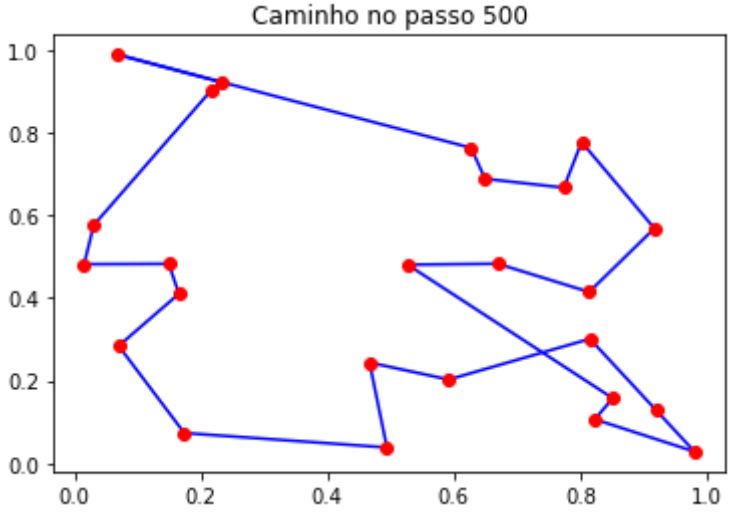

```
iteração: 300     energia/distância = 10.916309
```



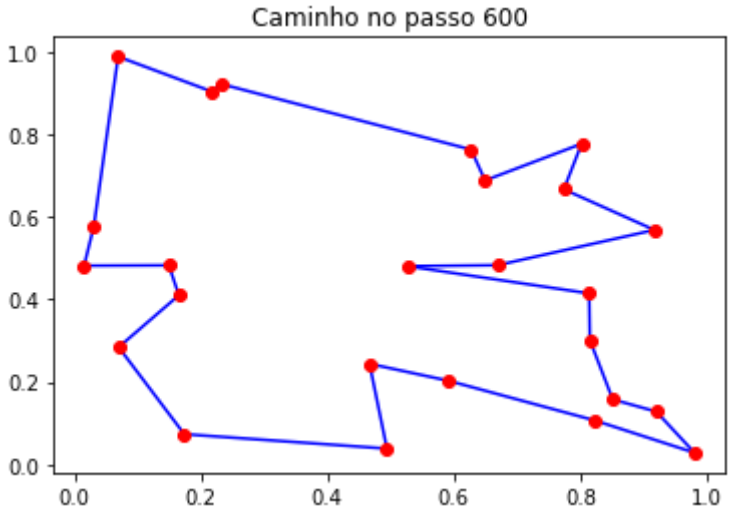

```
iteração: 400     energia/distância = 8.361478
```



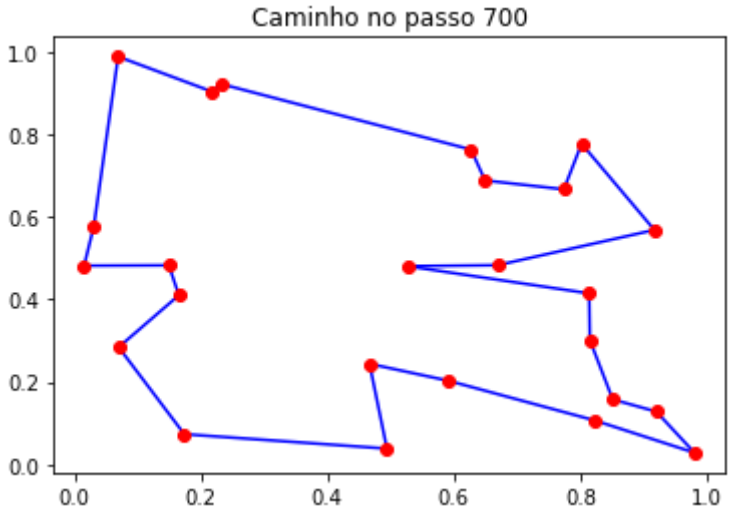

```
iteração: 500     energia/distância = 4.838454
```



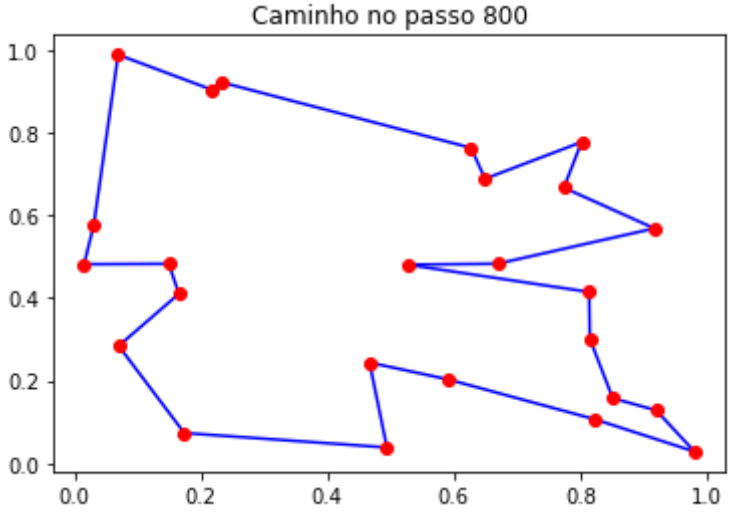

```
iteração: 600     energia/distância = 4.514074
```



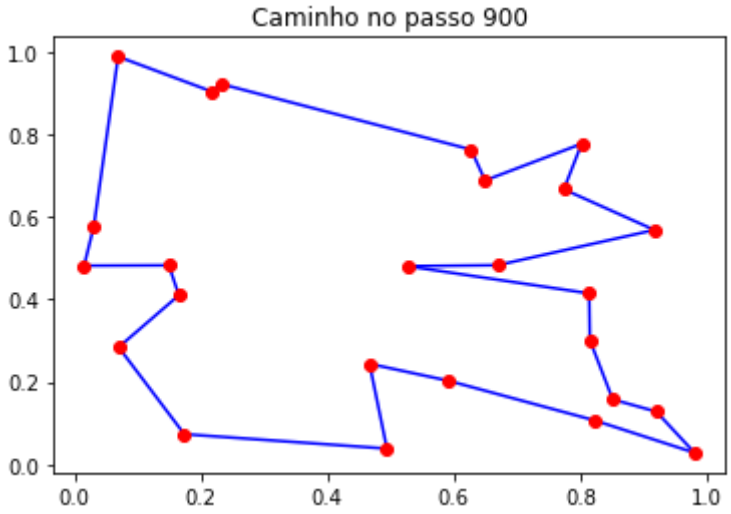

```
iteração: 700     energia/distância = 4.528400
```



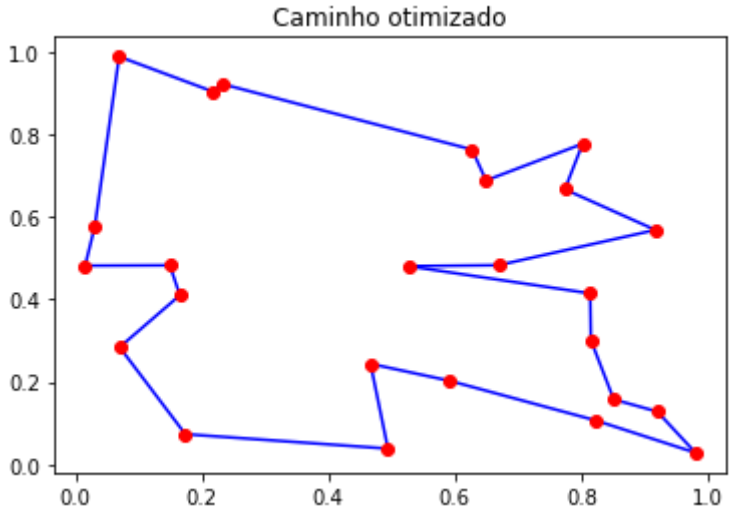

```
iteração: 800     energia/distância = 4.514074
```




```
iteração: 900     energia/distância = 4.514074
```




```
temperatura Limite na iteração 916
```

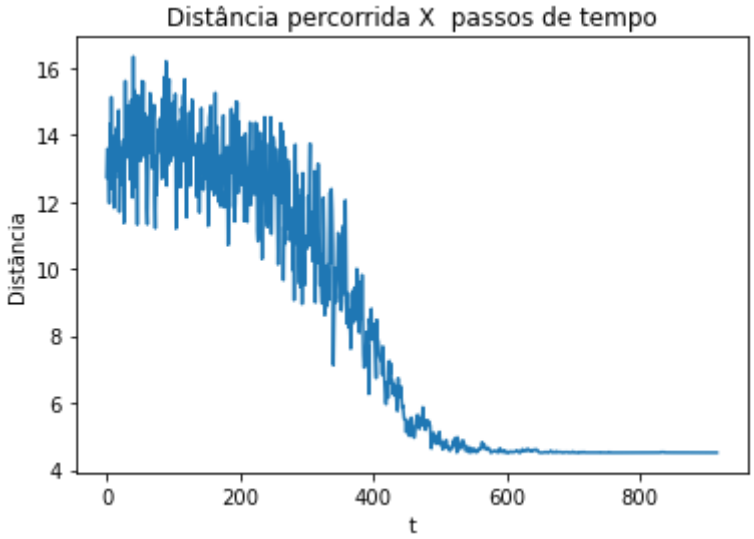


In [129…

```
plt.plot(dists)
plt.title(f"Distância percorrida X  passos de tempo")
plt.xlabel('t')
plt.ylabel('Distãncia')
```

Out[129…

```
Text(0, 0.5, 'Distãncia')
```



Pelo resultado acima, vê-se que o modelo alterna entre configurações intermediárias vez ou outra aceitando `"custos"` ou `energias/distâncias` maiores que a de passos anteriores. No passo 600 e no passo 700 por exemplp, ficou clara essa característica do método. Apesar disso, a grandeza em questão é otimizada, ou o mais próximo disso, atingindo um valor de `4.514074` em que pelo gráfico verifica-se que ocorreu por volta do passo 550.