Simulated_Annealing_IFEC

Alunos

André Luiz Moreira Dutra - 2019006345

Vitor Assunção Rabello de Oliveira - 2019007104

Vitor Rodarte Ricoy - 2019007112

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import exp
```

[2]:
```
def caminhadaInicial(N):
  cam = [i for i in range(N)]
  np.random.shuffle(cam)
  return cam

def calculaDist(N, x, y):
  dist = []
  for i in range(N):
    temp = []
    for j in range(N):
      temp.append(((x[i]-x[j])**2 + (y[i]-y[j])**2)**(1/2))
    dist.append(temp.copy())
  return dist

def calculaEnergia(cam, dist, N):
  ener = 0
  for i in range(N-1):
    ener += dist[cam[i]][cam[i+1]]
  ener += dist[cam[0]][cam[N-1]]
  return ener

def gerarNovaCaminhada(N, cam, dist):
  ncam = [0 for _ in range(N)]
  i=np.random.randint(0, N-1)
  j=i
  while j==i:
    j=np.random.randint(0, N-1)
```

```
  if i > j:
    ini = j
    fim = i
  else:
    ini = i
    fim = j
  for k in range(N):
    if k >= ini and k <= fim:
      ncam[k] = cam[fim-k+ini]
    else:
      ncam[k] = cam[k]
  esq = ini-1
  if esq < 0:
    esq = N-1
  dir = fim+1
  if dir > N-1:
    dir = 0
  de = -dist[cam[esq]][cam[ini]]-dist[cam[dir]][cam[fim]] +␣
 ↪dist[ncam[esq]][ncam[ini]] + dist[ncam[dir]][ncam[fim]]
  return ncam, de

def passoMonteCarlo(cam, dist, ener, t):
  N = len(cam)
  ncam, de = gerarNovaCaminhada(N, cam, dist)
  if de < 0 or np.random.random() < exp(-de/t):
    return ener+de, ncam
  return ener, cam
```

[3]:
```
#Implementação do metropolis por redução de temperatura
def algoritmoMetropolis(N, cam, dist, Ti, Tf, dt):
  energia = []
  ener = calculaEnergia(cam, dist, N)
  t = Ti
  counter = 0
  while t > Tf:
    ener, cam = passoMonteCarlo(cam, dist, ener, t)
    energia.append(ener)
    counter+=1
    if counter >= 100:
      t = t*dt
      counter = 0
  return energia, cam
```

[4]:
```
def plot_samples(N,Tis,Tfs,dts):
  fig, axs = plt.subplots(3,3,figsize = (20,15))
  x = [np.random.random() for _ in range(N)]
  y = [np.random.random() for _ in range(N)]
```

```
  dist = calculaDist(N, x, y)
  cam = caminhadaInicial(N)
  cams = [[[[]]*3]*3]*3
  for i,Ti in enumerate(Tis):
    for j,Tf in enumerate(Tfs):
      axs[i][j].set_title("Metropolis para $T_i$ = "+str(Ti)+", $T_f$ =␣
↪"+str(Tf))
      axs[i][j].set_xlabel("Iterações")
      axs[i][j].set_ylabel("Energia (distância total do caminho)")
      for k,dt in enumerate(dts):
        ener,c = algoritmoMetropolis(N,cam,dist,Ti,Tf,dt)
        cams[i][j][k] = c.copy()
        #print("Tf = "+str(Tf)+", dt = "+str(dt)+", energ = "+str(ener[-1]))
        axs[i][j].plot(ener,label = "dt = "+str(dt))
      axs[i][j].legend()
  fig.suptitle("Amostra de tamanho N = "+str(N))
  fig.show()
  return cams,x,y

def plot_cams(cams,x,y,Tis,Tfs,dts, N):
  fig, axs = plt.subplots(3,3,figsize = (20,15))
  for i,Ti in enumerate(Tis):
    for j,Tf in enumerate(Tfs):
      axs[i][j].set_title("Caminho Gerado pelo Metropolis para $T_i$ =␣
↪"+str(Ti)+", $T_f$ = "+str(Tf))
      for k,dt in enumerate(dts):
        c = cams[i][j][k]
        xC = [x[i] for i in c]
        xC.append(xC[0])
        yC = [y[i] for i in c]
        yC.append(yC[0])
        axs[i][j].plot(xC,yC,label = "dt = "+str(dt), marker = 'o')
      axs[i][j].legend()
  fig.suptitle("Amostra de tamanho N = "+str(N))
  fig.show()
```

Para que o algoritmo Metropolis possa ser executado, alguns parâmetros devem ser ajustados, sendo eles o número de vértices da amostra, a temperatura inicial, a temperatura final, e o fator de diminuição da temperatura. Cada um desses parâmetros pode afetar o algoritmo em algum aspecto, como o número de iterações feitas até a convergência, o valor da energia atingido após a convergência, ou a variação da energia ao longo dos passos do algoritmo.

Para observar os efeitos de cada um desses parâmetros no algoritmo, foram selecionados três valores para cada um deles e plotadas as progressões do valor da energia a cada passo do algoritmo para todas as combinações possíveis desses valores.

Abaixo estão os valores selecionados para cada parâmetro:

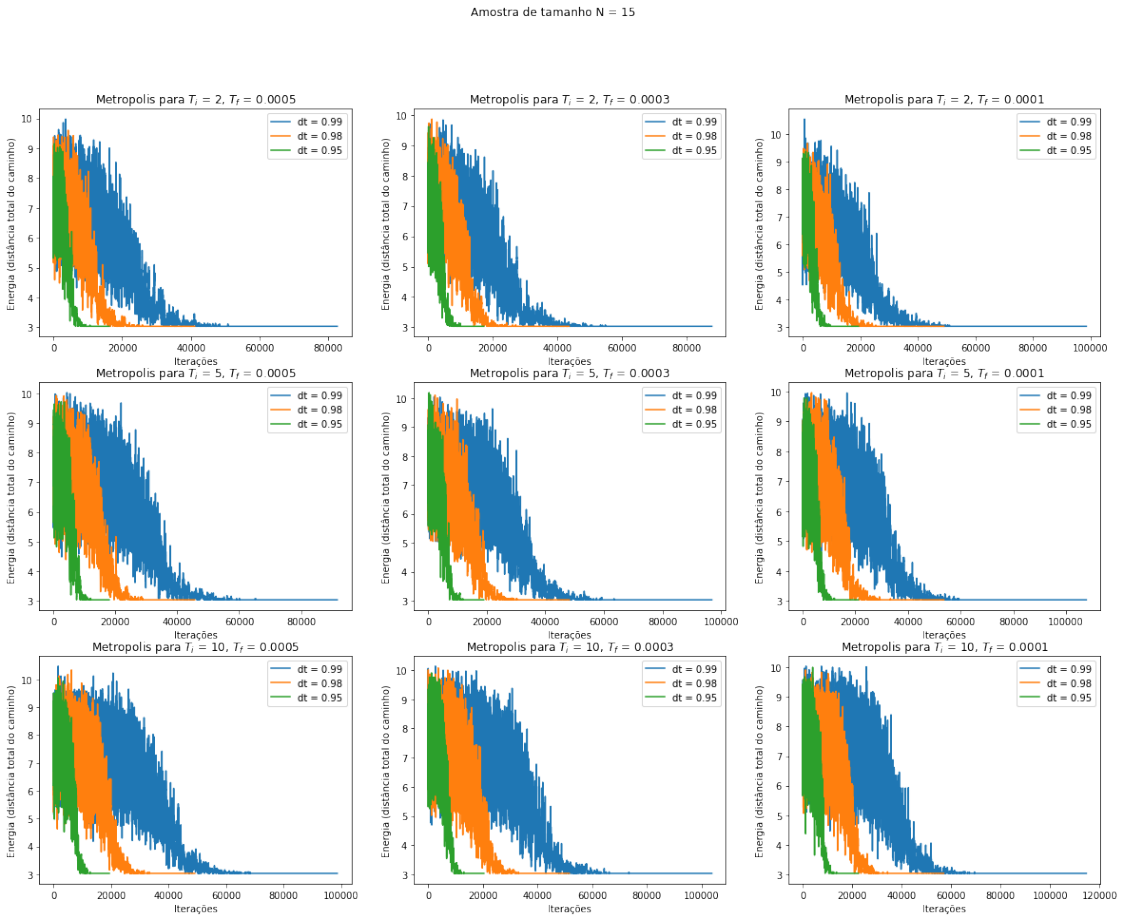
[5]:
```
Ns = [15, 30, 45]
Tis = [2, 5, 10]
Tfs = [0.0005, 0.0003, 0.0001]
dts = [0.99, 0.98, 0.95]
```

Os plots foram agrupados pelo tamanho da amostra. Para uma mesma amostra do tamanho selecionado, cada parâmetro restante foi variado, de modo que as colunas variam a temperatura inicial, as linhas variam a temperatura final e variações no fator da temperatura estão demonstradas em um mesmo plot.

Vejamos os resultados para a amostra de tamanho 15:

[6]:
```
cams,x,y = plot_samples(Ns[0],Tis,Tfs,dts)
```

[6]:



A mudança mais evidente com certeza se encontra na alteração do fator de diminuição da temperatura. Embora fosse esperado que o aumento do fator aumentasse o número de iterações, já que a temperatura demoraria mais para cair à temperatura final definida, alterações muito pequenas nesse valor produzem aumentos enormes no número de iterações, como podemos ver nos gráficos acima, nos quais um aumento de 0.04 no fator aumentou o número de iterações em cerca de 5 vezes. Isso provavelmente se dá pelo fato do fator ser multiplicado muitas vezes, ou seja, elevado

a uma potência alta, levando alterações em seu valor a causar efeitos exponenciais na temperatura.

Assim como o aumento do fator de diminuição da temperatura, também é esperado que o número de iterações aumente ao diminuir a temperatura final e aumentar a temperatura inicial. De fato, observando cada linha e cada coluna separadamente, essa diferença pode ser observada, embora seja mais sutil comparada à diferença trazida pelo fator de diminuição da temperatura.

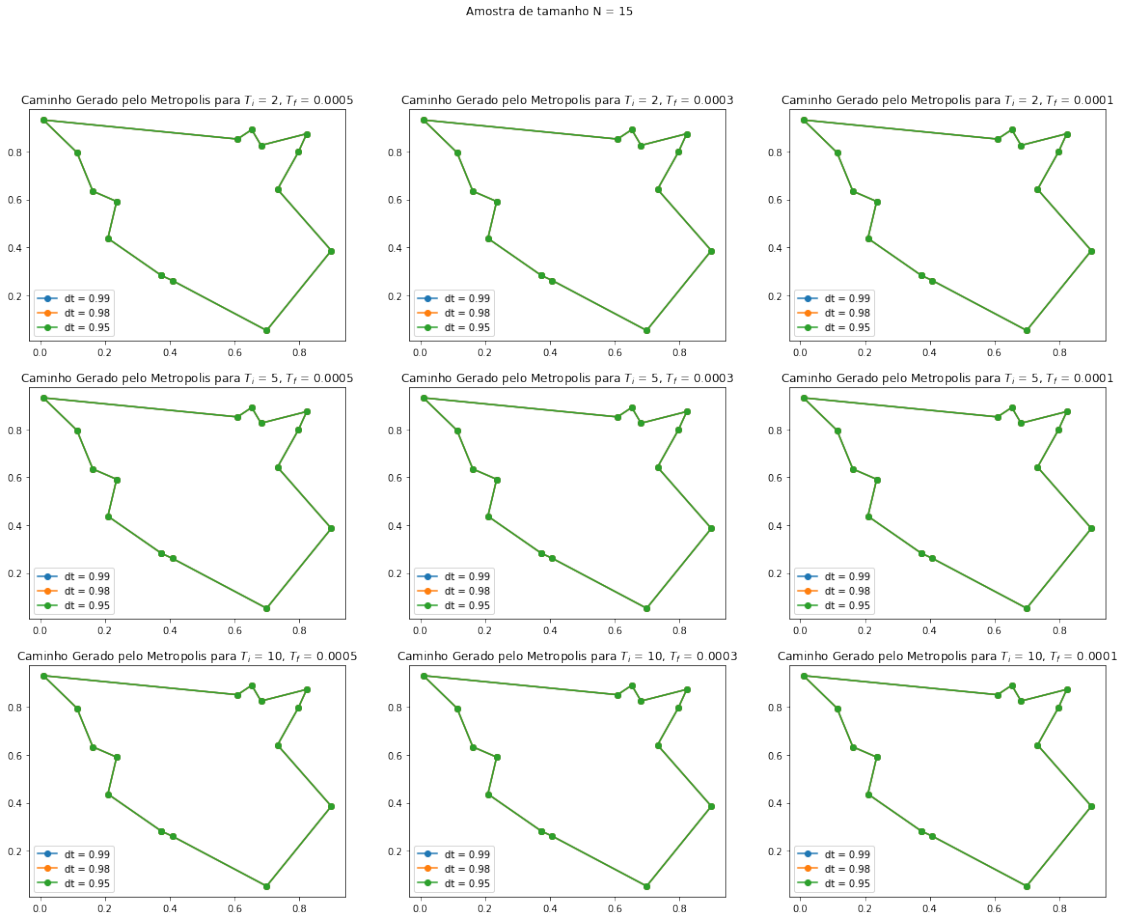
Aumentos de cerca de 4 na temperatura inicial causaram um aumento de cerca de 8% no número de iterações fixando os demais parâmetros. Diminuições na temperatura final também trouxeram aumentos de cerca de 8% no número de iterações, fixando os demais parâmetros, mas as alterações na temperatura final foram muito mais sutis, de cerca de 0.0002.

Apesar das diferenças no número de iterações, tanto a variação da energia ao longo do algoritmo quanto a energia final atingida após a convergência foi a mesma em todos os casos. Ao contrário do que se esperaria intuitivamente, o aumento no número de iterações não levou o algoritmo, nesse tamanho de amostra, a convergir para energias menores.

Podemos também observar os diferentes caminhos gerados por essas execuções:

[7]:
```
plot_cams(cams,x,y,Tis,Tfs,dts, Ns[0])
```

[7]:

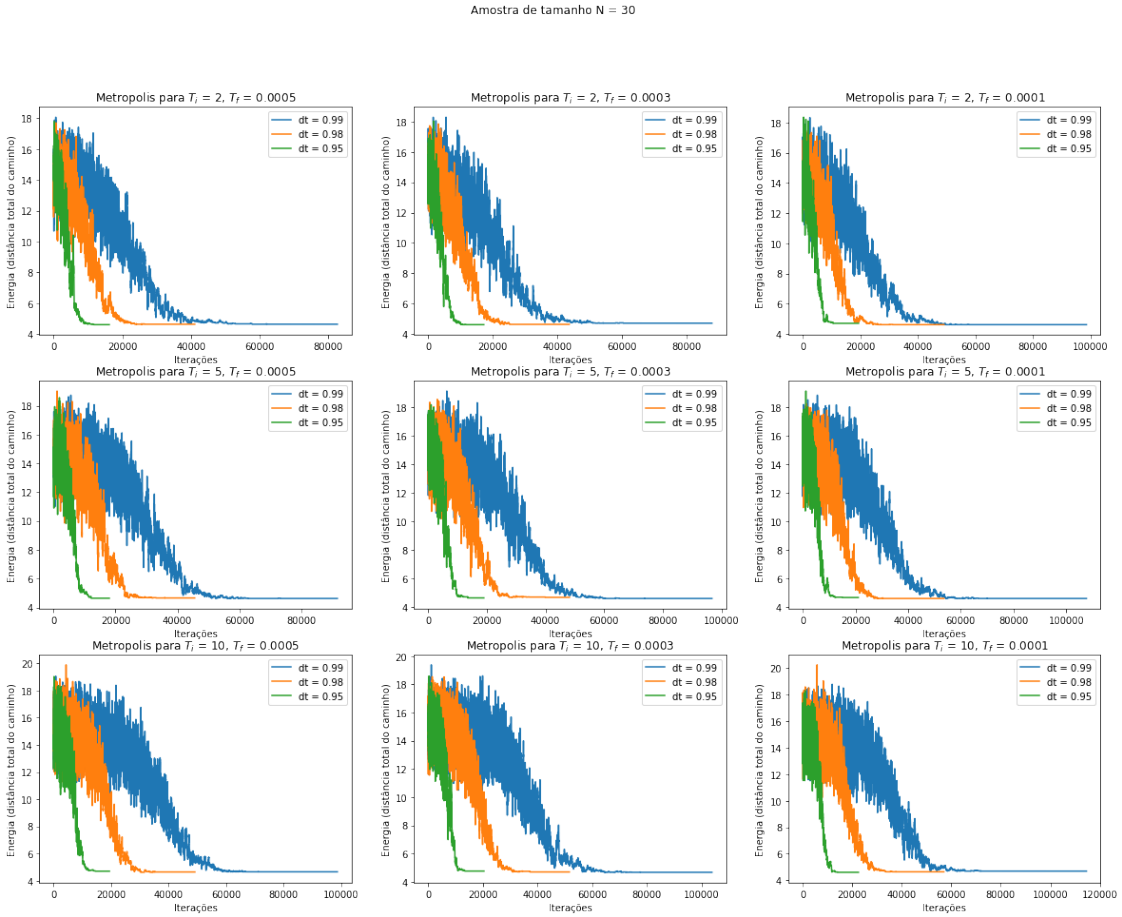


Podemos ver que as soluções encontradas são idênticas. Isso condiz com o que foi encontrado pelos gráficos: a energia final parece ser a mesma para todos os cenários, portanto as soluções têm um custo muito próximo entre si.

Observemos agora as instanciações do algoritmo para amostras de tamanho maior:

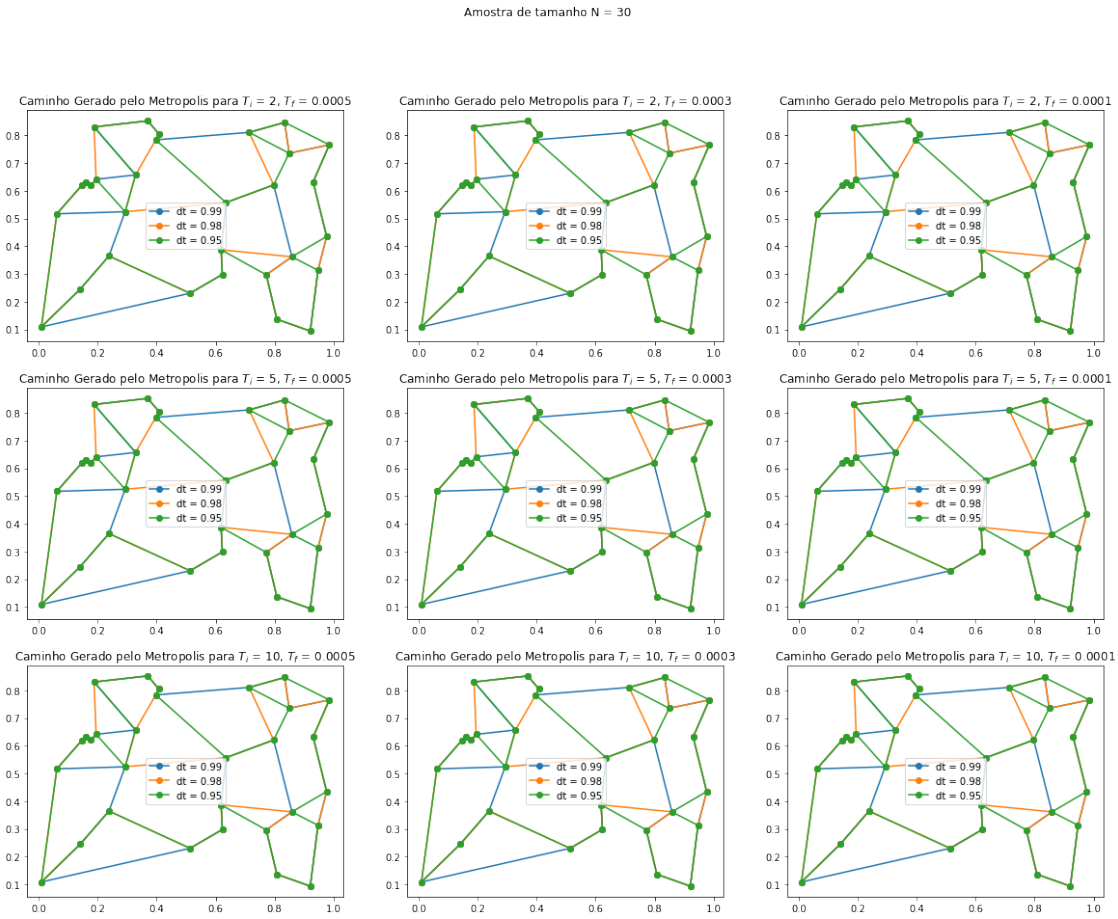
[8]:
```
cams, x, y = plot_samples(Ns[1],Tis,Tfs,dts)
```

[8]:



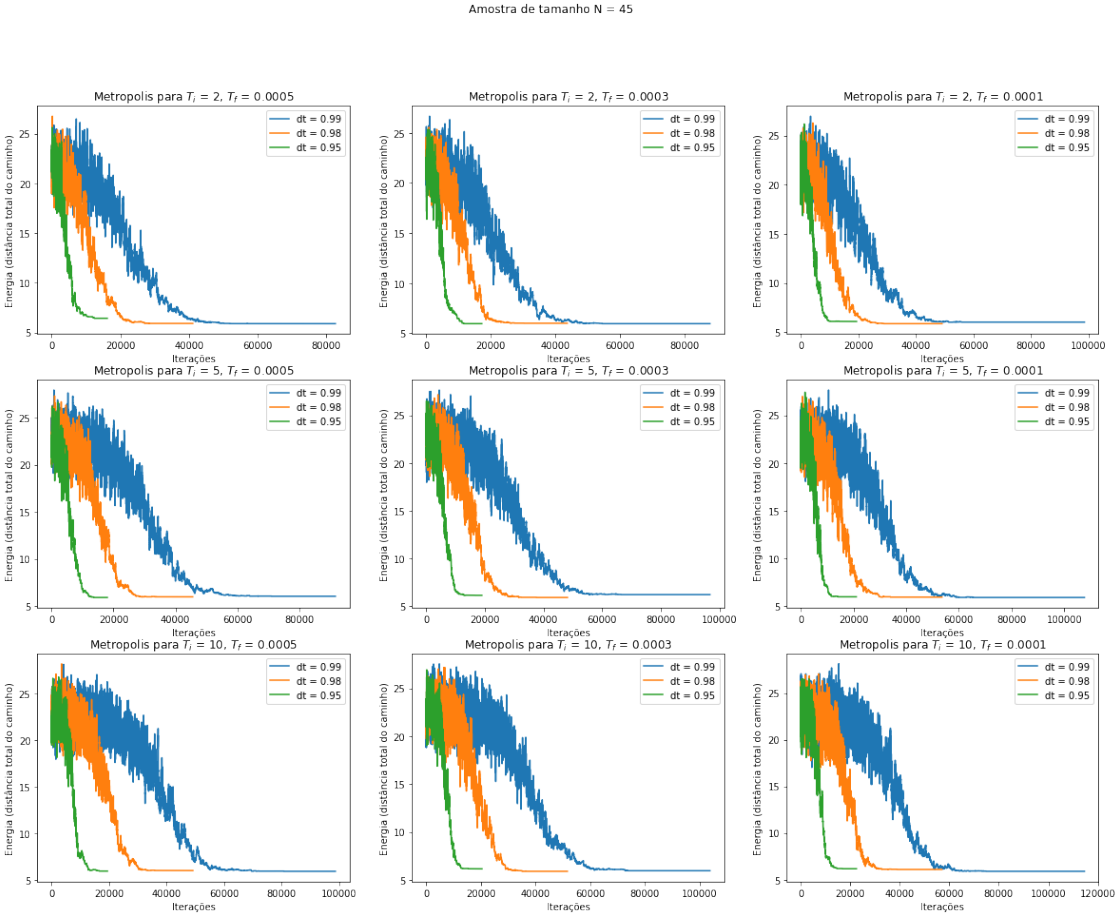
[9]:
```
plot_cams(cams,x,y,Tis,Tfs,dts, Ns[1])
```

[9]:



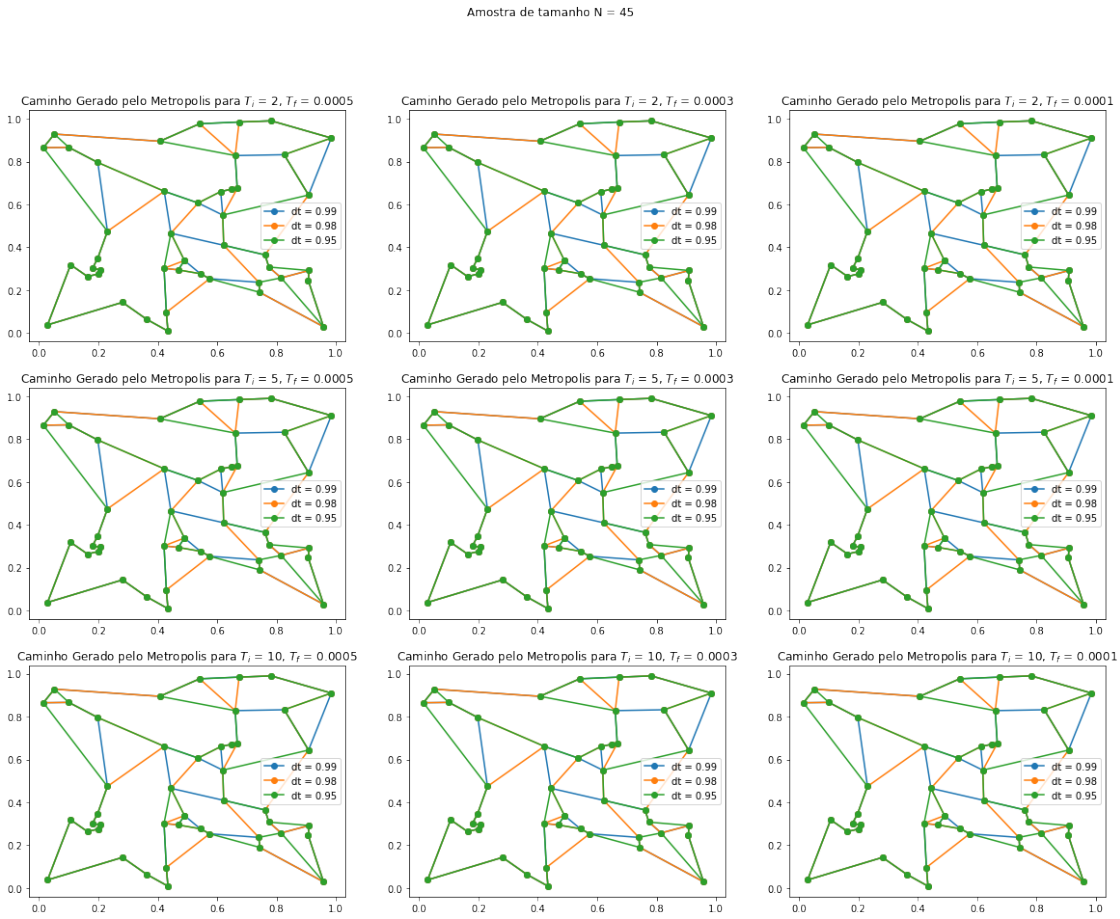
[10]:
```
cams, x, y = plot_samples(Ns[2],Tis,Tfs,dts)
```

[10]:



[11]:
```
plot_cams(cams,x,y,Tis,Tfs,dts, Ns[2])
```

[11]:



As mesmas observações feitas anteriormente se mantêm para amostras de tamanhos maiores. Fica evidente, no entanto, que aumentando a amostra a variação da energia em cada instância diminui. Os gráficos de instâncias do algoritmo executadas com amostras maiores são menos espessos, do início ao ponto de convergência. Muito provavelmente isso se dá porque em uma amostra maior há mais opções de trechos pequenos para serem escolhidos, na etapa de inversão de um trecho aleatório do passo do Monte Carlo. Assim, conjectura-se que haveria mais chance de que uma inversão de um trecho pequeno fosse feita, não afetando tanto a distância total e, assim, diminuindo a variação da energia.

Além disso, em amostras maiores começamos a ver casos em que instâncias convergem para valores diferentes, como podemos ver em algumas instâncias para a amostra de tamanho 45. No entanto, essa diferença é minúscula, de centésimos, quase imperceptível, portanto um resultado satisfatório é atingido mesmo que os valores em instâncias sejam diferentes, o que é confirmado pelos caminhos encontrados.

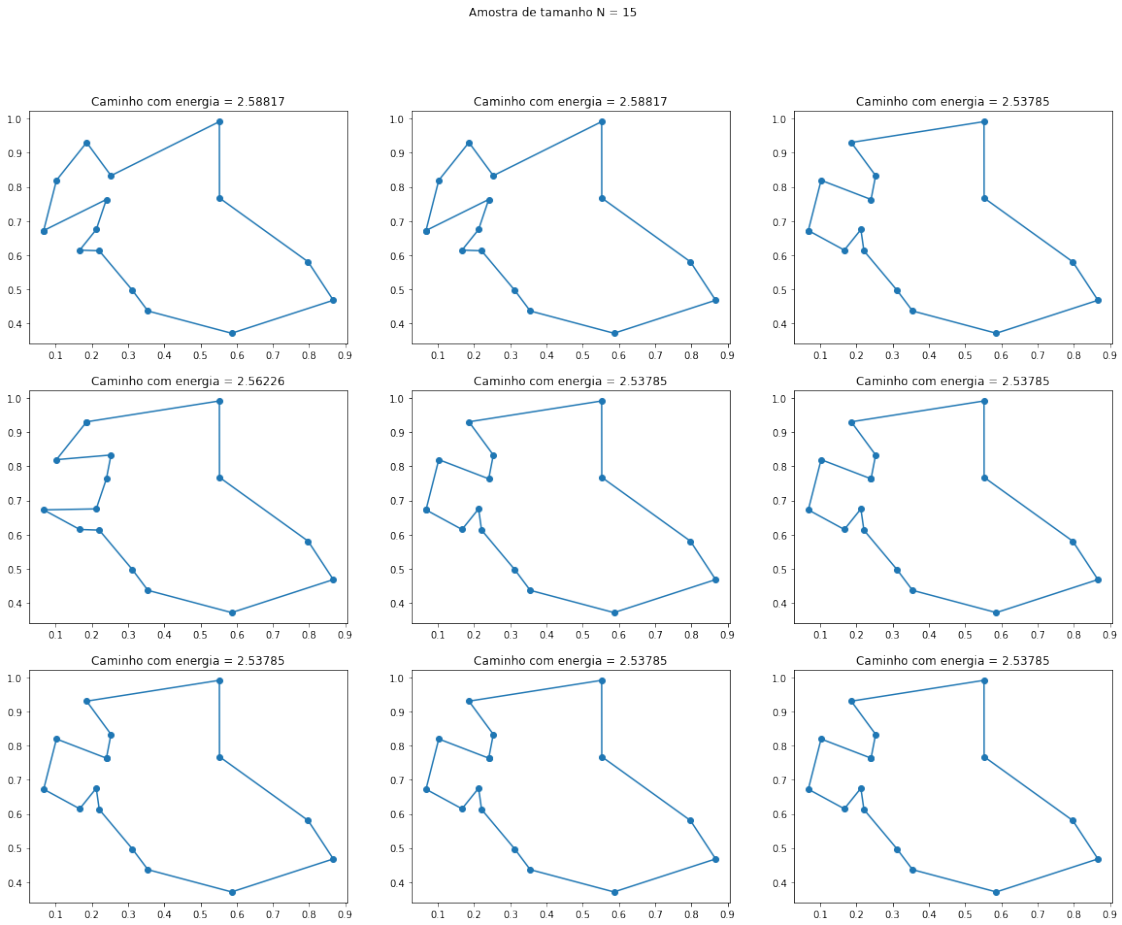
Também, aparentemente o caminho final encontrado do algoritmo não recebe tanta influência dos valores de T_i e T_f, enquanto os valores de dt geram diferenças iguais e consistentes entre os diferentes parâmetros T_i e T_f. Também observamos pequenas variações nos caminhos encontrados para os valores diferentes de dt. Essas variações são bem pequenas e refletam a diferença minúscula da energia final encontrada ao analisar o gráfico de energia.

Podemos concluir que o melhor caminho a ser percorrido parece ser, no geral, os caminhos com o parâmetro $dt = 0.98$, pois, em média, eles possuem custos menores. Finalmente, plotamos os caminhos gerados utilizando os mesmos parâmetros para o algoritmo, em execuções diferentes, com o objetivo de analisar como o caminho gerado muda de acordo com a aleatoridade do algoritmo. Serão analisadas 9 execuções com $N = 15, 30, 45$. Também utilizaremos $T_i = 5$, $T_f = 0.0003$ e $dt = 0.98$:

[13]:
```
def plot_cam_diferentes_execucoes(N):
  fig, axs = plt.subplots(3,3,figsize = (20,15))
  x = [np.random.random() for _ in range(N)]
  y = [np.random.random() for _ in range(N)]
  dist = calculaDist(N, x, y)
  cam = caminhadaInicial(N)
  for i in range(3):
    for j in range(3):
      ener,c = algoritmoMetropolis(N,cam,dist,5,0.0003,0.98)
      axs[i][j].set_title("Caminho com energia = "+str(round(ener[-1], 5)))
      xC = [x[i] for i in c]
      xC.append(xC[0])
      yC = [y[i] for i in c]
      yC.append(yC[0])
      axs[i][j].plot(xC,yC,marker = 'o')
  fig.suptitle("Amostra de tamanho N = "+str(N))
  fig.show()

plot_cam_diferentes_execucoes(Ns[0])
```

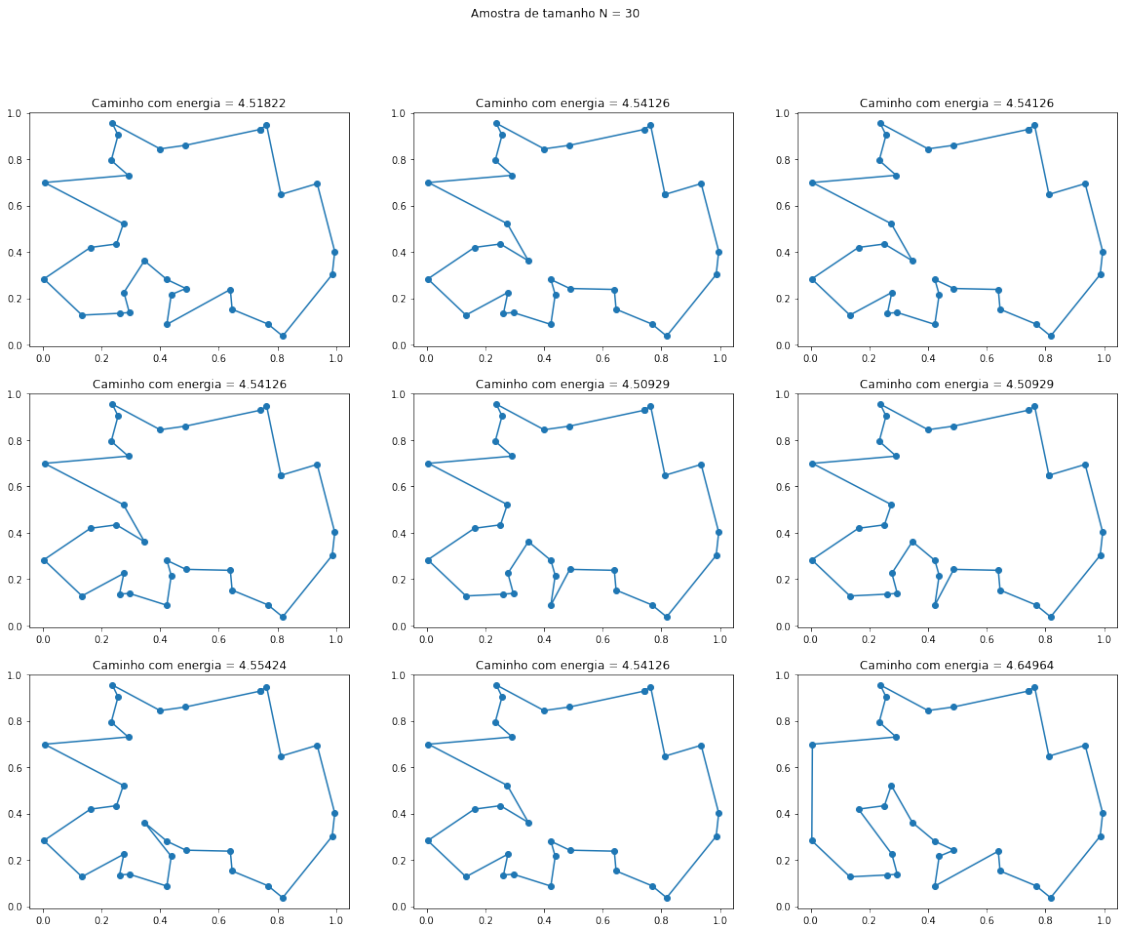
[13]:



Para $N = 15$ vemos que o algoritmo é bem estável, gerando soluções muito parecidas e com o valor de energia bem próximos, tendo uma diferença de energia de aproximadamente 0.05, entre a solução de maior e menor energia.

[14]:
```
plot_cam_diferentes_execucoes(Ns[1])
```

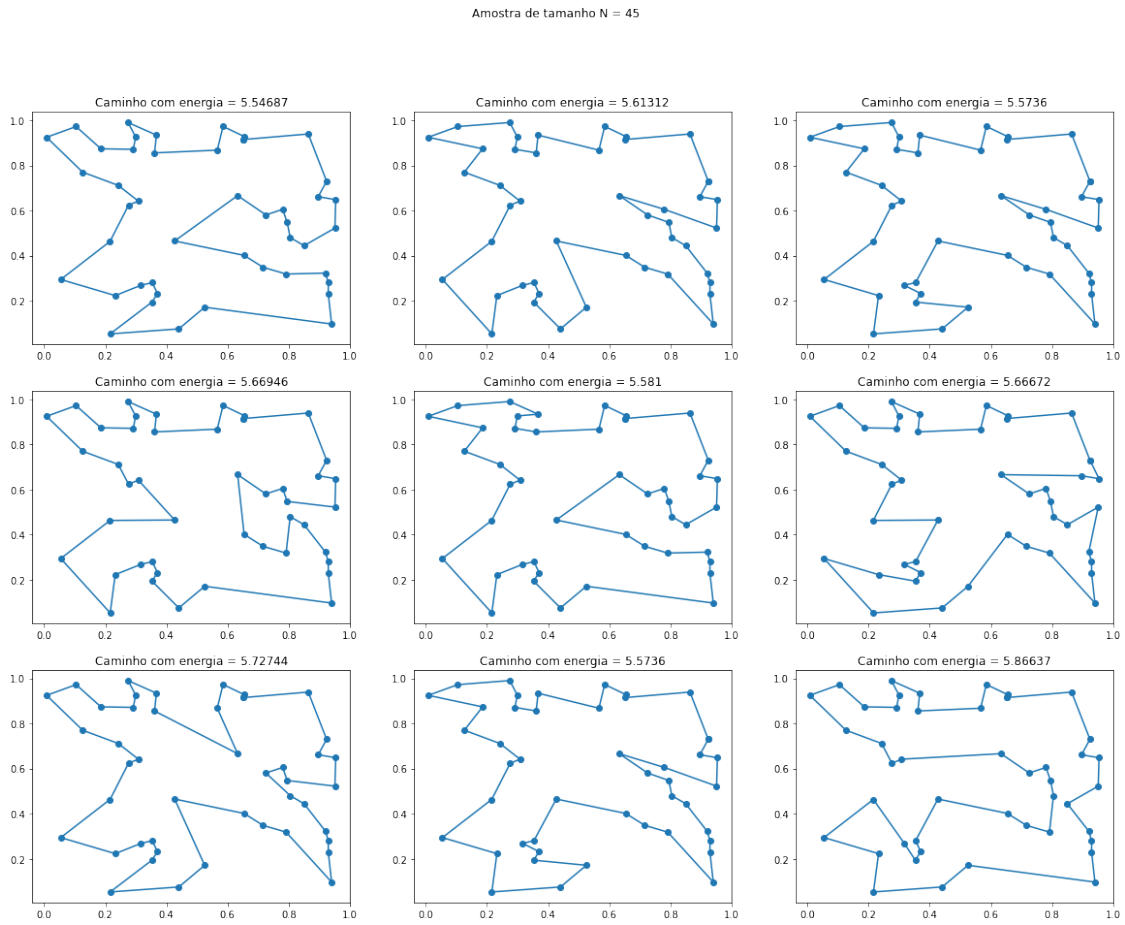
[14]:



Para $N = 30$ observamos uma instabilidade maior, tendo caminhos mais diferentes visualmente e valores de energia também mais diferentes entre si, tendo uma diferença de energia de aproximadamente 0.13, entre a solução de maior e menor energia.

[15]:
```
plot_cam_diferentes_execucoes(Ns[2])
```

[15]:



Finalmente, para $N = 45$ observamos uma instabilidade ainda maior, tendo caminhos bem diferentes visualmente e valores de energia também bem diferentes entre si, tendo uma diferença de energia de aproximadamente 0.32, entre a solução de maior e menor energia.