THE TRAVELING SALESMAN PROBLEM

Francisco Jesus-89084-50% Gonçalo Freixinho-89251-50%

UNIVERSIDADE DE AVEIRO LEI

7 de Novembro de 2018

Introdução

O problema do caixeiro vianjante pode ser descrito da seguinte maneira: dadas m cidades, se o caixeiro vianjante deseja passar por cada uma delas uma única vez e voltar à cidade de ínicio, qual é o caminho mais curto que pode percorrer?

Este problema tem sido alvo de grande atenção por parte de matemático e de cientistas da computação. Isto deve-se ao facto de este problema representa um grupo maior de problemas conhecidos como problemas de otimização combinatória. Se uma solução mais eficiente fosse encontrada para resolver este problema, então seria possível desenvolver algoritmos mais eficientes para outros problemas semelhantes.

Métodos de resolução

O método usado para calcular o caminho mais curto para cada um dos conjuntos de cidades foi calcular a distância entre cada cidade e somá-la a uma variável que guardava a distância do total do percurso. Depois compara-se com a variável min_length, se a distância total fôr menor que min_length, min_length fica igual à distância total. Para a distãncia máxima do percurso a método é o mesmo que o anterior, diferindo só na confição if, que verica se max_length é menor que a distância total, se fôr verdadeiro, max_length fica igual à distância total.

Para realizar o histograma adicionou-se um array de inteiros de tamanho 10000 como varável global, que durante a execução da função tsp,quando o número de cidades fôr igual a 12 ou 15, qualquer distância total que seja calculada será passada como indíce do array e incrementa o valor do número nessa posição do array por 1.

Finalmente, para determinar mostrar o caminho mais curto e o caminho mais longo, utilizam-se os arrays max_tour e min_tour. Durante o processo de determinar a distância mínima/máxima, dentro da condição if faz-se um ciclo for para copiar os valores do array “a” para as matrizes mencionadas anteriormente.

Conclusões

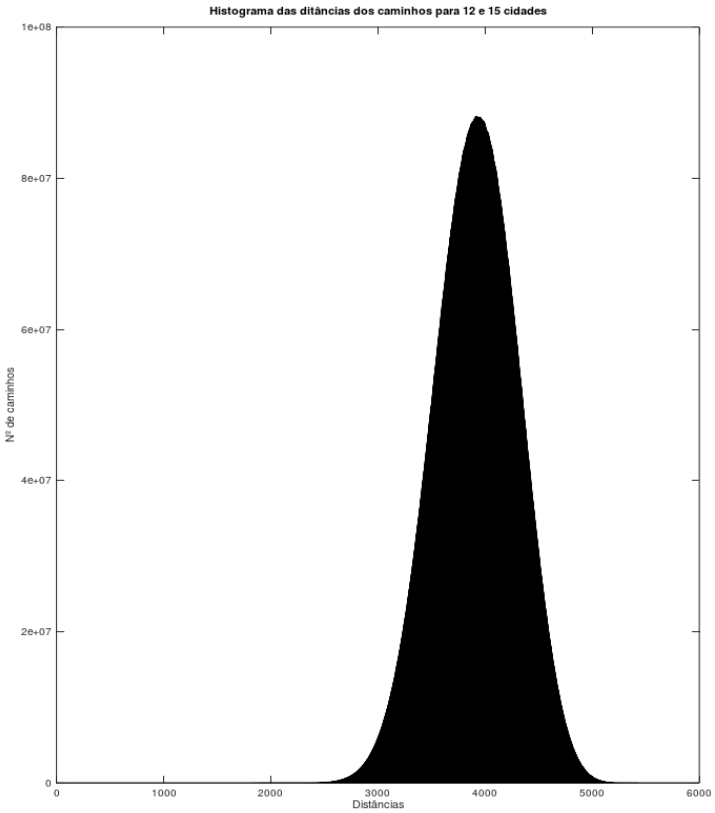

Fig1-Histograma das distâncias para 12 e 15 cidades

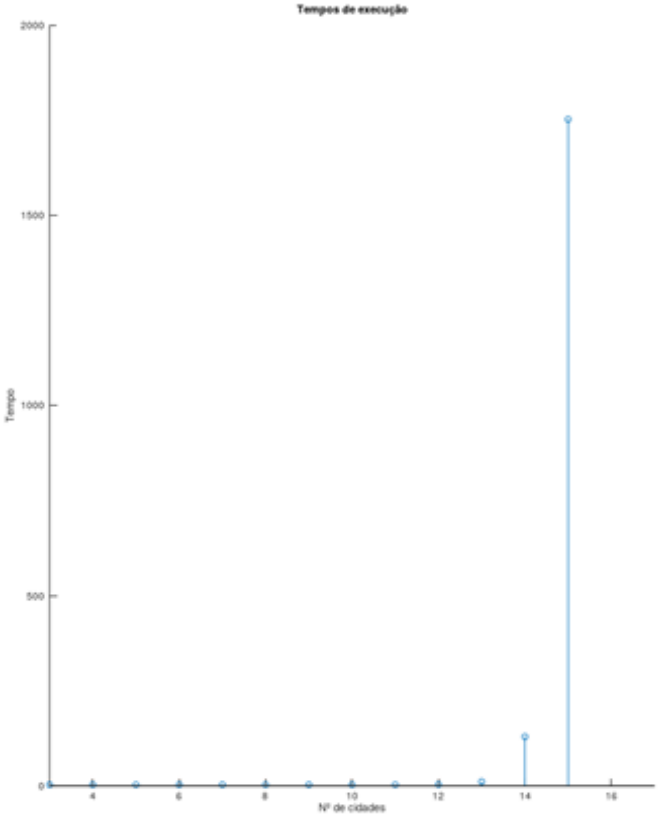

Fig2-Tempos de execução consoante o número de cidades

Começando a analisar o primeiro gráfico, é possível observar uma distribuição normal das distâncias com uma média por volta dos 4000km, isto significa que uma grande parte dos percursos criados têm uma distância próxima deste valor e só poucos percursos se mantêm nos valores extremos.

Quanto ao gráfico que representa os tempos de execução, os tempos de execução apresentam um crescimento exponencial que se torna bastante notório quando se passa das 14 para as 15 cidades e o programa começa a ficar bastante lento quando mais cidades se adicionam aos percursos. Com isto pode-se afirmar que este método de resolução é pouco eficiente para resolver este problema.

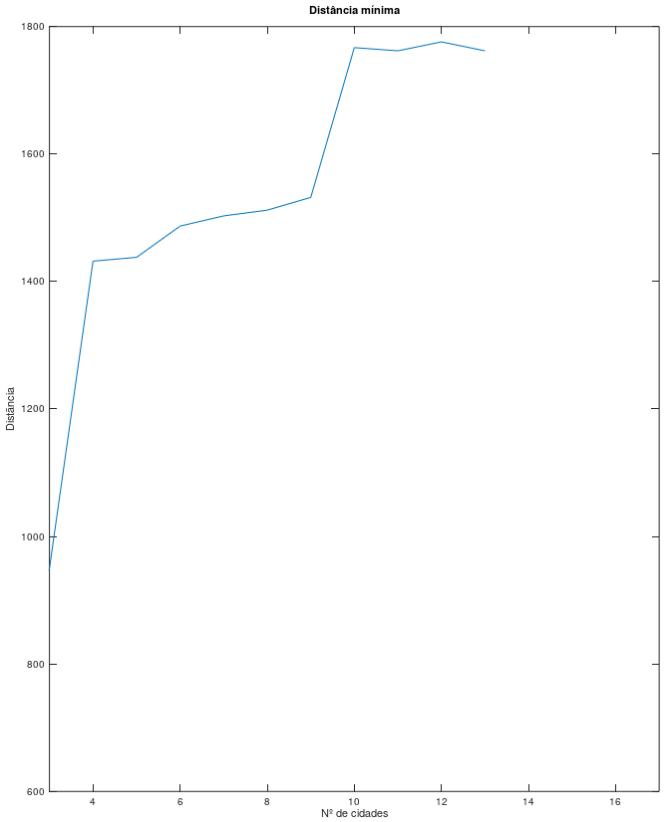

Fig3-Distância mínima consoante o número de cidades

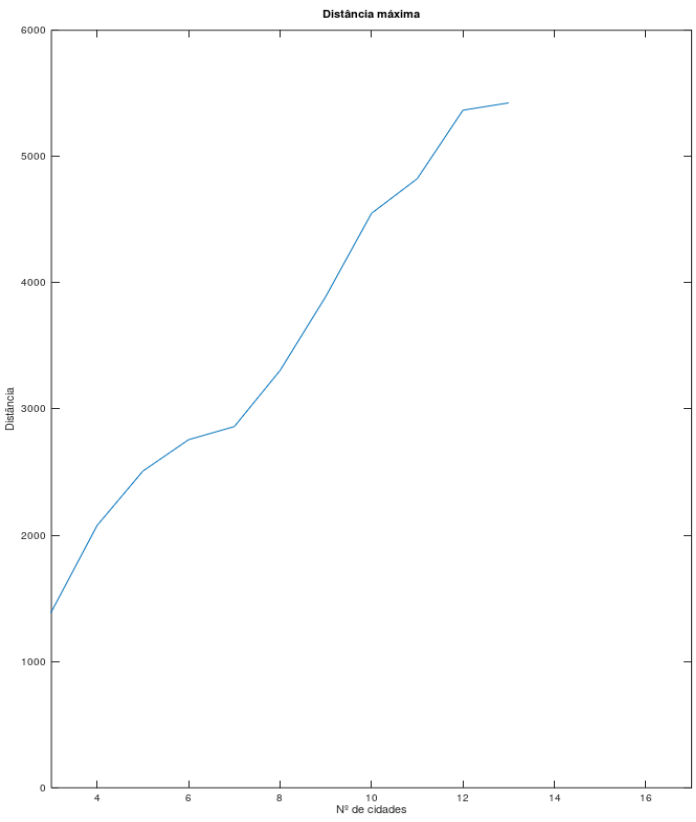

Fig4-Distância máxima dependendo do número de cidades

Considerando agora os gráficos referentes às distâncias máximas e mínimas (com special=0) de cada conjunto de cidades. O gráfico das distâncias mínimas apresenta duas variações elevadas e dois períodos em que se mantém quase constante, isto pode-se dever à adição de uma cidade mais distante do conjunto de cidades anterior.

Quanto ao gráfico das distâncias máximas, este segue um crescimento linear, sempre que se adiciona uma cidade a distância aumenta, como já se esperava.

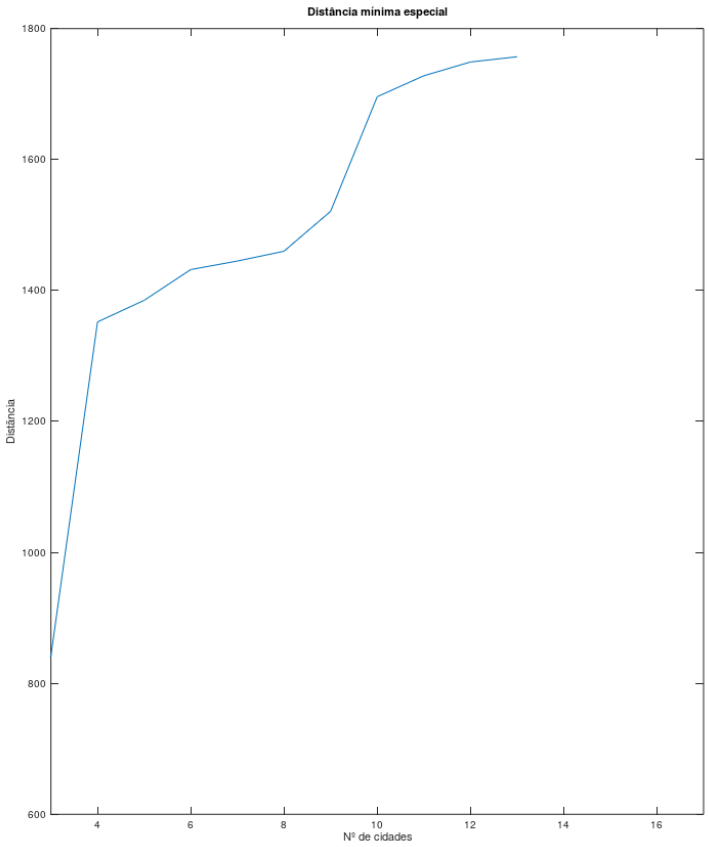

Fig5-Distância mínima consoante o número de cidades com special=1

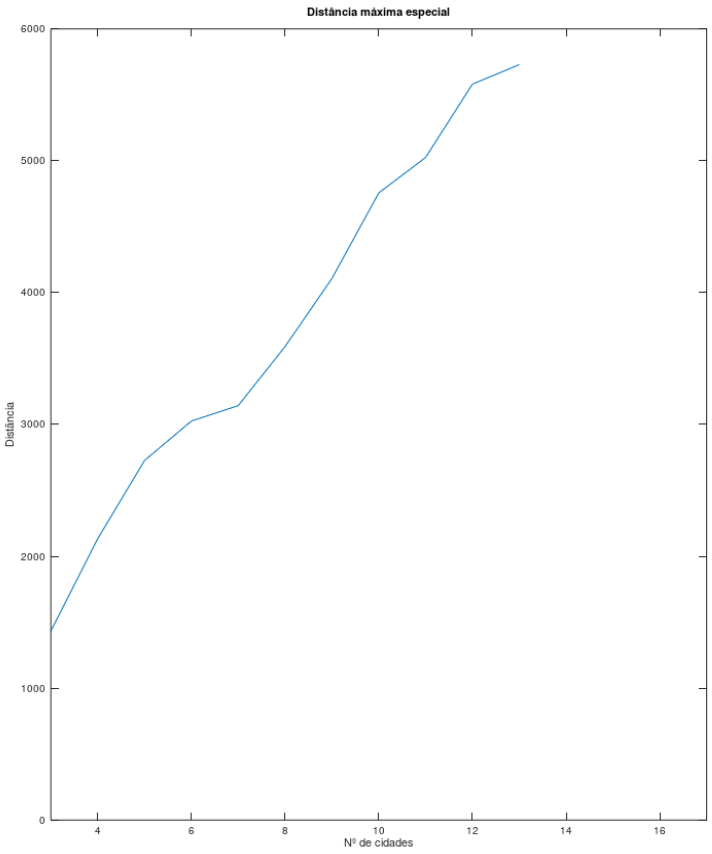

Fig6-Distância máxima dependendo do número de ciades com special=1

Os gráficos das distâncias mínimas e máximas (com special=1) apresentam o mesmo comportamento dos anteriores, sendo que agora existe alterações em cada distância, devido ao facto que agora a distância entre as cidades é assimétrica.

Bibliografia

Hoffman K.L., Padberg M., Rinaldi G. (2013) Traveling Salesman Problem. In: Gass S.I., Fu M.C. (eds) Encyclopedia of Operations Research and Management Science. Springer, Boston, MA

PROBLEMA DO CAIXEIRO-VIAJANTE aberto. In: Wikipédia: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Problema_do_caixeiro-viajante> Acesso em: 7 nov 2018

```c
//
// Francisco Jesus
// Student name
// ...
//
// AED, 2018/2019
//
// solution of the traveling salesman problem
//

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/resource.h>

#include "cities.h"
#include "../P01/elapsed_time.h"

//
// record best solutions
//

static int min_length,max_length;
static int min_tour[max_n_cities + 1],max_tour[max_n_cities + 1];
static long n_tours;
const int maxSize = 10000;
int *hist;


//
// first solution (brute force, distance computed at the end, compute best and worst tours)
//


void tsp_v1(int n,int m,int *a)
{
         int i, t, dis;
         int sumDis = 0;

  if(m < n - 1)
    for(i = m;i < n;i++)
    {
      t = a[m];
      a[m] = a[i];
      a[i] = t;
      tsp_v1(n,m + 1,a);
      t = a[m];
      a[m] = a[i];
      a[i] = t;
    }
  else
  { // visit permutation
    n_tours++;
    // modify the following code to do your stuff
         for (i = 0; i < n; i++) {


                  if (i >= 1) {
                           dis = cities[a[i - 1]].distance[a[i]];
                           sumDis += dis;
                  }
         }
         sumDis += cities[a[n - 1]].distance[0]; //para voltar ao inicio


         if (n == 12 || n == 15) {// array com valores do histograma
                  hist[sumDis]++;

         }

         if (sumDis <= min_length) {
                  min_length = sumDis;
                  for (i = 0; i < n; i++) {
                           min_tour[i] = a[i];
                  }
         }
         if (sumDis >= max_length) {
                  max_length = sumDis;
                  for (i = 0; i < n; i++) {
                           max_tour[i] = a[i];
                  }
         }
  }
```

```
}


//
// main program
//

int main(int argc,char **argv)
{
  hist = malloc(sizeof(int*) * maxSize);
  if (!hist) { perror("malloc arr"); exit(EXIT_FAILURE); };


  int n_mec,special,n,i,a[max_n_cities];
  char file_name[32];
  double dt1;
  int n_cities = 15;
  n_mec = 89084; // CHANGE THIS!
  special = 1;
  init_cities_data(n_mec,special);
  printf("data for init_cities_data(%d,%d)\n",n_mec,special);
  fflush(stdout);
#if 0
  print_distances();
#endif
  for(n = 3;n <= n_cities;n++)
  {
    //
    // try tsp_v1
    //

    dt1 = -1.0;
    if(n <= 16)
    {
      (void)elapsed_time();
      for(i = 0;i < n;i++)
        a[i] = i;
      min_length = 1000000000;
      max_length = 0;
      n_tours = 0l;
      tsp_v1(n,1,a); // no need to change the starting city, as we are making a tour
      dt1 = elapsed_time();

      printf("tsp_v1(%d) finished in %8.3fs (%ld tours generated)\n",n,dt1,n_tours);
      printf("  min %5d [",min_length);
      for(i = 0;i < n;i++)
        printf("%2d%s",min_tour[i],(i == n - 1) ? "]\n" : ",");
      printf("  max %5d [",max_length);
      for(i = 0;i < n;i++)
        printf("%2d%s",max_tour[i],(i == n - 1) ? "]\n" : ",");

      fflush(stdout);

      if(argc == 2 && strcmp(argv[1],"-f") == 0)
      {
        min_tour[n] = -1;
        sprintf(file_name,"min_%02d.svg",n);
        make_map(file_name,min_tour);
        max_tour[n] = -1;
        sprintf(file_name,"max_%02d.svg",n);
        make_map(file_name,max_tour);
      }
    }
  }
  /*
  for (i = 0; i < maxSize; i++) {

          printf(" %d%s", hist[i],(i==maxSize-1)? "\n" : " ");
  }
  */
  free(hist);

  return 0;
}
```