The Traveling Salesman Problem

and

Similar Problems

By

Abderrahim Hechachena

27/05/2018

abderrahimhechachena@yahoo.co.uk

In this papers, I will discuss the nature of the TSP and similar problems to provide an algorithm for approximate solution. This set of problems is defined by a matrix of weights, that could be, distances between cities to be minimized or expenses, associated with assignments of jobs to be minimized or profit to be maximized. This set of problems fall into two categories.

- 1) The Traveling salesman problem is a symmetric or non symmetric matrix, with the diagonal elements set to a maximum value to prohibit selection, as these cells are distances between cities themselves. it is not possible to say the distance between city a and city a is something different from zero. The diagonal could be set to zero to find a maximum value or to a maximal value to find a minimum assignment.
- 2) The assignment problem, is a square matrix with no conditions imposed on the diagonal as the rows and columns are independent. in this case we could have a set of companies as column names and jobs as row names, the cells contain costs. We want to use all companies to do all jobs at the same time, so we want to minimize the cost.

In each case of the above, using a weighting matrix, we should have only one cell for each column and one cell for each row, with the sum equal to a target maximum or minimum. Lets start with the diagonal elements as our first choice. Given this situation we can not change any single cell or any collection of odd cells, 1,3,5.. cells at one time, as this will disturb the condition, we set, to have one cell only for each column and one cell only for each row. so we are going to start with a couple of cells from the assignment and check if the opposite cells to this couple have a sum less than or greater than the target cells, for example: cell m[1,1] = 1000 and cell m[2,2] = 1000 so the total is 2000, but cell m[1,2] = 7 and cell m[2,1] = 3. then we can swap the first couple with the second couple and reduce the value of the assignment. We can proceed in this way till we have no possible swap that minimize or maximize the target assignment.

Is this enough?

Swapping a couple of cells may provide a good approximation or even a complete solution, but that may not be enough in some cases. The proof is in this example. suppose we reached an approximate solution, swapping couples with couples only. The solution is 'a' that contains the couples (a1,a3), (a4,a2), (a1,a2), (a3,a4) with the opposite couples 'not chosen' (b1,B1), (b2,B2), (b3,B3), (b4,B4) with sum a1 + a2 + a3 + a4 > b1 + b2 + b3 + b4. Now, we can swap the first four with the last four and obtain a better solution. It was not possible to do so using couples only, because, the large values B1, B2, B3, B4 prevented couples swap. This will mean any algorithm designed to find the optimal solution, may need to check more than simple couples. Couples may provide a good approximation though. As there is no obstacle for any cell to be included using other cells with relatively low value, the probability of such case happening is very small.

TSP solution, R example

The code bellow could do all types of approximation for these problems. the weight matrix could be, non square matrix, but it should be squared by adding the right values to avoid selection of non existent values. This will help get partial solutions to those problems.

```
> ################### minimizing expression
> tspMin = expression({
+   for(i in 1:(n-1)){
+     for(j in (i+1):n){
+       A = m[r[i],c[i]]
+       B = m[r[j],c[j]]
+       A1 = m[r[i],c[j]]
+       B1 = m[r[j],c[i]]
```

```
+       if((A1 + B1)<(A + B)){
+         ci = c[i];
+         c[i] = c[j]
+         c[j] = ci
+         flag = 1
+       }
+     }
+   }
+ })
> ##################### maximizing expression
> tspMax = expression({
+   for(i in 1:(n-1)){
+     for(j in (i+1):n){
+       A = m[r[i],c[i]]
+       B = m[r[j],c[j]]
+       A1 = m[r[i],c[j]]
+       B1 = m[r[j],c[i]]
+       if((A1 + B1)>(A + B)){
+         ci = c[i];
+         c[i] = c[j]
+         c[j] = ci
+         flag = 1
+       }
+     }
+   }
+ })
> ##################### main function
> # m is the input matrix, M have values 0,1,2,3 as shown bellow
> TSP = function(m, M){
+   c = NULL
+   if(M > 3){print(paste(M, ': is not a legal value;
+                         shoose values from 0,1,2,3 only',sep = ''))}else{
+   L = max(m)
+   n = dim(m)[1]
+   if(M==0){mod = tspMin;diag(m)=n*L}
+   if(M==1){mod = tspMax;diag(m)=0}
+   if(M==2){mod = tspMin}
+   if(M==3){mod = tspMax}
+
+   n = dim(m)[1]
+   c = 1:n
+   r = 1:n
+   flag = 1
+   while(flag == 1){
+     flag = 0
+     eval(mod)
```

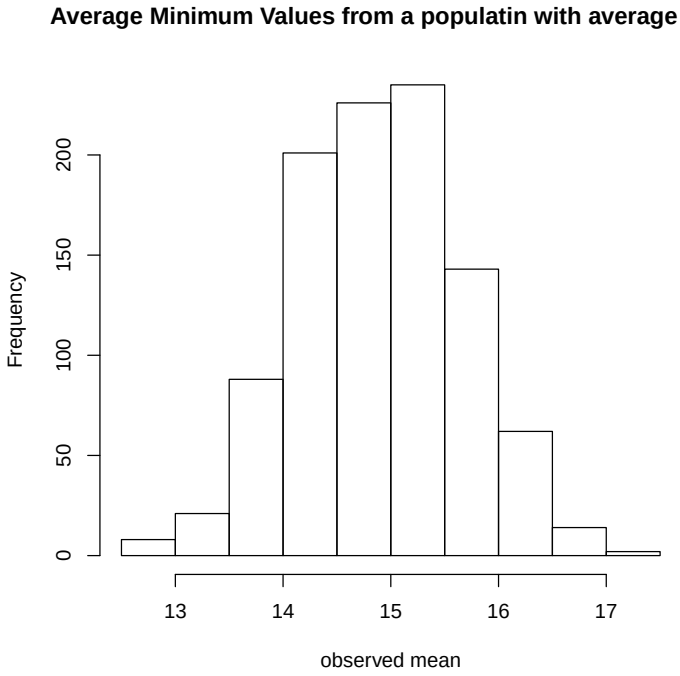
```
+   }
+   }
+  c
+ }
> ###################### exploring all four situations
> # generate a matrix n by n using a poisson process with lambda = L
> n = 10;L = 20;sampleSize = 1000
> m = matrix(rpois(n*n,L),n,n)
> m

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]   17   19   22   30   21   20   13   18   21    18
 [2,]   13   20   17   20   22   17   22   13   19     8
 [3,]   18   17   22   18   14   18   17   21   16    15
 [4,]   22   21   22   22   25   19   17   22   17    17
 [5,]   30   17   24   19   24   29   26   13   15    23
 [6,]   19   20   27   15   17   16   20   17   17    21
 [7,]   24   30   26   27   16   21   28   22   22    17
 [8,]   20   24   18   16   21   19   18   29   18    20
 [9,]   20   18   20   28   23   16   22   21   13    15
[10,]   22   18   24   28   18   25   21   16   22    16

> # the output is the set of columns reorderd, rows will preserve their order 1..n
> # option 0 for minimizing without using the diagonal elements
> TSP(m,0)

 [1]  7 10  1  9  8  4  5  3  6  2

> # option 1 for maximizing without using the diagonal elements
> TSP(m,1)

 [1]  9  7  8  5  1  3  2 10  4  6

> # option 2 for minimizing including diagonal elements
> TSP(m,2)

 [1]  7 10  1  6  8  4  5  3  9  2

> # option 3 for maximizing including diagonal elements
> TSP(m,3)

 [1]  9  7  3  5  1 10  2  8  4  6

> # output averages
> c = TSP(m,0)
> a = 0;for(i in 1:n){a = a + m[i,c[i]]};a/n

[1] 15.2
```

```
> c = TSP(m,1)
> a = 0;for(i in 1:n){a = a + m[i,c[i]]};a/n

[1] 24.9

> c = TSP(m,2)
> a = 0;for(i in 1:n){a = a + m[i,c[i]]};a/n

[1] 15.1

> c = TSP(m,3)
> a = 0;for(i in 1:n){a = a + m[i,c[i]]};a/n

[1] 25.3
```

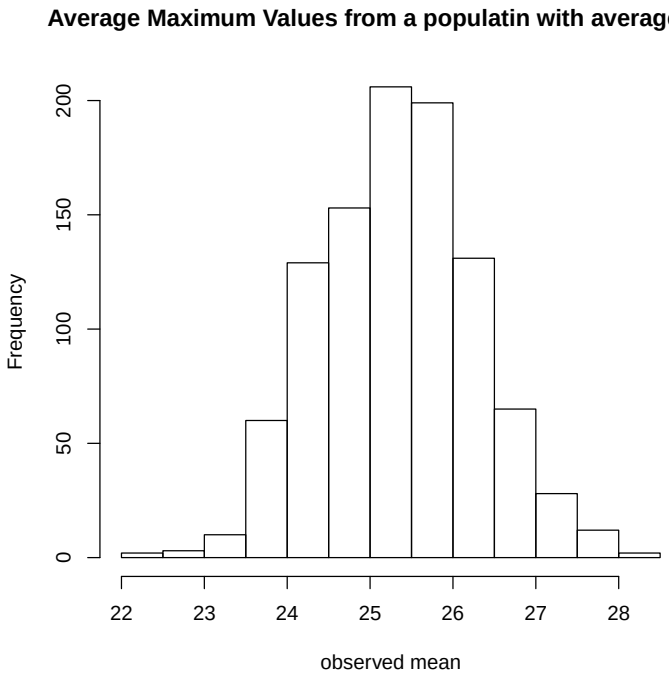
The distribution of the mean

with a sample size = 1000

```
> ### minimus sample s1
> s1 = c()
> for(i in 1:sampleSize){
+ m = matrix(rpois(n*n,L),n,n)
+   c = TSP(m,0)
+   a1 = 0;for(i in 1:n){a1 = a1 + m[i,c[i]]}
+   s1 = c(s1,a1)
+ }

> hist(s1/n ,main =
+        'Average Minimum Values from a populatin with average = 20' ,
+      xlab = 'observed mean')
```




```
> ## maximum sample s2
> s2 = c()
> for(i in 1:sampleSize){
+ m = matrix(rpois(n*n,L),n,n)
+   c = TSP(m,1)
+   a2 = 0;for(i in 1:n){a2 = a2 + m[i,c[i]]}
+   s2 = c(s2,a2)
+ }

> hist(s2/n,main =
+        'Average Maximum Values from a populatin with average = 20',
+      xlab = 'observed mean')
```



From these histograms we see that the mean of the minimized solution is well bellow the distribution average 20. The mean of the maximum solutions is well above the average 20 as well. assuming that the total of every solution is normally distributed according to CLT, s1 and s2 each contain 1000 normally distributed observations. We calculate the p value for both minimum and maximum solutions as fellows.

```
> sum(s1);S1 = sum(((s1-mean(s1))/sd(s1))^2)
[1] 149508
> dchisq(S1,sampleSize-2)
[1] 0.008916894
> sum(s2);S2 = sum(((s2-mean(s2))/sd(s2))^2)
[1] 253886
> dchisq(S2,sampleSize-2)
[1] 0.008916894
```

It is clear that both solutions reject the null hypothesis at a highly significant level. These statistical results proves the goodness of these solutions.