

Traveling Salesman Problem Heuristics

Arman Boyacı

October 26, 2007

1 Introduction

In this study, some heuristic methods to solve traveling salesman problem will be analyzed. Those heuristics, **nearest neighbor**, **arbitrary insertion** and **farthest insertion**, are called classical construction heuristics. Besides, an improvement heuristic, actually well known and most used one, which is called 2-opt will be applied to this three construction heuristic methods.

Our aim is making some comparisons among those three construction heuristic and one improvement heuristic. In this manner, three (small, medium and large) type of instances are selected from TSPLib on the net. Each implementation of three construction heuristics will be initiate 10 times and then each of them will be followed by the improvement heuristic. The starting point which is a parameter for our heuristics will be changed before each run. However same parameters will be used in order to have comparable results for heuristics.

Every heuristic will be assessed according to three criteria *solution quality*, *robustness* and *time efficiency*. The results in detail as well as the implementation codes of each heuristic could be found in Appendix.

2 Data Analysis

2.1 Solution Quality

In most cases, solution quality is measured as the deviation from the optimal solution. In our case optimal solution are provided from TSPLib. Final results of solution quality for the construction heuristics and then followed by the improvement heuristic are shown respectively on Table 1 and Table 2.

	Instance Size					
	Small		Medium		Large	
	Objective Value	Deviation	Objective Value	Deviation	Objective Value	Deviation
Optimum Value	15780	-	6773	-	50801	-
Nearest Neighbor	19244	22%	8520	26%	61667	21%
Arbitrary Insertion	17590	11%	8605	27%	63024	24%
Farthest Insertion	17180	8%	8124	20%	62515	23%

Table 1: Average objective values obtained by using each classic heuristic

2.2 Robustness

There can be many different criteria when we talk about the robustness. Here in our study we consider robustness of objective value when we change the parameter, the starting point. The results given in Table 3 are standard deviations of objective values obtained by applying each heuristic to three different sizes of instances.

	Instance Size					
	Small		Medium		Large	
	Objective Value	Deviation	Objective Value	Deviation	Objective Value	Deviation
Optimum Solution	15780	-	6773	-	50801	-
Nearest Neighbor	16205	3%	7165	6%	53098	5%
Arbitrary Insertion	16646	4%	7711	11%	56281	11%
Farthest Insertion	16469	4%	7549	11%	56243	11%

Table 2: Average objective values obtained by using each heuristic followed by 2-opt improvement heuristic

	Instance Size					
	Small		Medium		Large	
		2-opt		2-opt		2-opt
Nearest Neighbor	878	174	149	42	1861	263
Arbitrary Insertion	172	82	55	76	87	502
Farthest Insertion	526	121	68	79	434	284

Table 3: Standart deviations of each heuristic methods

2.3 Time Efficiency

It is clear that the evaluation of time efficiency is not easy. It can be depend on implementation as well as the hardware we use. However in our case, four heuristic has been tested on the same machine. Due to this reason, Table 4 could be used to compare them among each other. The numbers may not be so important but the ratios can be considerable.

3 Conclusion

In this section, the results obtained will be discussed. First of all, we can start with the solution quality. If we consider only the construction heuristics, among three of them, farthest insertion algorithm give us the best values. For example, for small instances we have only a 8% gap between the optimum value. However this gap increase rapidly respect to the instance size. On the hand, we could observe that 2-opt algorithm works very well with the nearest-neighbor algorithm. The gap between the optimum value decreases to 3% for small instances, 5% even for larger instances. Secondly, computational time and the robustness are also important. Since nearest neighbor algorithm followed by 2-opt method took significantly more time than farthest insertion algorithm and moreover deviations between results obtained by nearest neighbor are clearly higher. Another observation that we should make that using the improvement heuristic 2-opt algorithm took a lot of time! We should use it in the cases where we need absolutely good quality solutions.

All in all, nearest neighbor algorithm must be always following an improvement heuristic. It took times on the hand we get good results. If computational time is important and only one of the construction heuristic will be selected, that must definitely be the farthest insertion method.

	Instance Size					
	Small		Medium		Large	
		2-opt		2-opt		2-opt
Nearest Neighbor	0,15	5	4	125	40	600
Arbitrary Insertion	0,4	4	6	110	50	550
Farthest Insertion	0,4	5	8	110	78	416

Table 4: Computational times of each heuristic methods

4 Appendix

Results In Detail

Matlab Codes

Nearest Neighbor

```
tic
mevcut = baslangic_sehri;
optimumtur(1) = mevcut;
D(:,mevcut) = [];
k = 2;
while ( k ~= length(C{1})+1 )
    j = bul_enkucukj(D,mevcut);
    minimumj = D(1,j);
    optimumtur(k) = minimumj;
    mevcut = minimumj;
    D(:,j) = [];
    k = k + 1;
end

optimumtur = [ optimumtur baslangic_sehri ];

toc

tic
% Maliyet hesapla
for (i=1:length(optimumtur)-1)
    toplammesafe = E(optimumtur(i),optimumtur(i+1)) + toplammesafe;
end
toc
```

Arbitrary Insertion

```
% Birinci iterasyon
mevcut = baslangic_sehri;
optimumtur(1) = mevcut;
optimumtur(2) = mevcut;
D(:,mevcut) = [];

tic
% Diger iterasyonlar
while ( length(optimumtur) ~= size(D,1))
    mevcut = bul_randomnokta(D);
    ekle = D(1,mevcut);
    optimumtur = insertion(optimumtur, ekle, bul_enkucukmaliyetk(optimumtur,E, ekle ));
    D(:,mevcut) = [];
end
toc

tic
% Maliyet Hesapla
for (i=1:length(optimumtur)-1)
    toplammesafe = E(optimumtur(i),optimumtur(i+1)) + toplammesafe;
end
toc
```

Farthest Insertion

```
% Birinci iterasyon
mevcut = baslangic_sehri;
optimumtur(1) = mevcut;
optimumtur(2) = mevcut;
D(:,mevcut) = [];

tic
% Diger iterasyonlar
```

```

while ( length(optimumtur) ~= size(D,1))
    mevcut = bul_turaenuzaknokta(optimumtur, D);
    ekle = D(1,mevcut);
    optimumtur = insertion(optimumtur, ekle, bul_enukucukmaliyetk(optimumtur,E, ekle ));
    D(:,mevcut) = [];
end
toc

tic
for (i=1:length(optimumtur)-1)
    toplammesafe = E(optimumtur(i),optimumtur(i+1)) + toplammesafe;
end
toc

```

2-opt

```

iyilestirilmis_sonmesafe = 0;
iyilestirilmis_tur = optimumtur;
enbuyukkazanc = 1;
m = 0;

kazanclar = [];
sonuclar = [];

tic
while ( (enbuyukkazanc > 0) )
    enbuyukkazanc = 0;
    a = 0;
    b = 0;
    for (i=1:length(iyilestirilmis_tur)-2)
        for (j=i+2:length(iyilestirilmis_tur)-1)
            kazanc = - E( iyilestirilmis_tur(i),iyilestirilmis_tur(j) ) - E(iyilestirilmis_tur(i+1),iyilestirilmis_tur(j))
            if (kazanc > enbuyukkazanc)
                a = i;
                b = j;
                enbuyukkazanc = kazanc;
            end
        end
    end
    if(a~=0)
        m = m+1;
        iyilestirilmis_tur(a+1:b) = iyilestirilmis_tur(b:-1:a+1);

        iyilestirilmis_mesafe = 0;
        for (i=1:length(iyilestirilmis_tur)-1)
            iyilestirilmis_mesafe = E(iyilestirilmis_tur(i),iyilestirilmis_tur(i+1)) + iyilestirilmis_mesafe;
        end
        kazanclar(m) = enbuyukkazanc;
        sonuclar(m) = iyilestirilmis_mesafe;
    end
end
toc

for (i=1:length(iyilestirilmis_tur)-1)
    iyilestirilmis_sonmesafe = E(iyilestirilmis_tur(i),iyilestirilmis_tur(i+1)) + iyilestirilmis_sonmesafe;
end

```