AN ALTERNATIVE ALGORITHM TO CREATE A ROUTE OPTIMIZED FOR THE TRAVELING SALESMAN PROBLEM

Edimar Veríssimo da Silva

INPE, ARTIFICIAL INTELLIGENCE - CAP-354
São José dos Campos, SP
yugi386@yahoo.com.br
May, 2017

ABSTRACT

This article presents the problem of the symmetrical traveling salesman (the distance between the city A → B is the same distance as the city B → A) and a non-deterministic algorithm to solve it in some cases using time and feasible computational resources.

Keywords: traveling salesman, combinatorial analysis, graphs.

I. INTRODUCTION

This work deals with the classic problem of the symmetrical traveling salesman (henceforth PCV). In item II we present the problem and verify that it is a NP-complete problem, where we easily have a combinatorial explosion that makes difficult the calculation of routes for a large number of cities. In item III we discuss some questions regarding the algorithm we propose to find solutions for the PCV. Such questions involve the computer memory limit and processing speed issues. In item IV we present our algorithm and show numerically how it reduces the search space to avoid combinatorial explosion. In item V we show some processing results of the algorithm in tests for sets from 5 to 12,500 cities. Due to computer memory exhaustion problems we were unable to test the algorithm for an even larger number of cities.

Finally we conclude, according to observations made in the tests, that our algorithm does have some effectiveness, although it cannot be compared with specialized algorithms that find shorter routes to this problem.

II. UNDERSTANDING THE PROBLEM

The problem of the traveling salesman is one of the most studied problems of combinatorial optimization. It can be paraphrased in the following situation: given a set of cities C, and the distances between them, find a route R that, starting from a given city A, passes through all the cities of C, only once in each one of them, and returns to city A (point of origin) making the shortest or least costly path [1].

The origin of the problem of the traveling salesman (PCV) dates back to the 1800s, the problem having been defined from the year 1920 onwards. Since then several challenges have been proposed and many solved (without necessarily finding the optimum solution for a large number of cities). Among the applications of this problem are the manufacture of electronic circuit boards, task sequencing, robot control, vehicle routing, etc [4].

Although the problem is easy to understand, it is NP-complete and grows exponentially. For a small number of cities the response can be exact and fast but for a large set of cities there is no deterministic algorithm that provides the optimum response (the best route, the shortest path) in a reasonable time [2].

> *"A search problem is NP-complete if all search problems are reduced to it. This is a very strong demand! For a problem to be NP-complete, it must be able to solve all the world's search problems [2]."*

To illustrate the difficulty of enumeration of the possible routes in the PCV we see that there are (n-1)! possibilities, where n represents the number of cities. For small numbers this calculation is quick but for large numbers the method of looking for the best solution by checking all possible routes is not feasible. The fact that the calculation of combinations is based on the factorial function makes the combinatorial explosion reach very quickly.

Table 1 illustrates these values:

NUMBER OF CITIES (N)	POSSIBILITIES (N-1)!
8	5.040
10	362.880
11	3.628.800
12	39916..800
30	$8{,}84176 \times 10^{30}$
50	$6{,}08281 \times 10^{62}$
1.000	$4{,}02387 \times 10^{2564}$

Table 1: Route Combinations

To get an idea of how large the number of route combinations is for 1000 cities it is enough to say that the number of atoms in the observable universe is something around 1080 [5]. This number is infinitely smaller than the number of route combinations for 1000 cities. Thus, the exhaustive search to solve the PCV for this number of cities is unthinkable. Really good alternatives, but that do not guarantee the optimal solution, are approximate algorithms, use of heuristics, genetic algorithms, among others [4].

> *"One way to reduce the complexity in solving the problem computationally is through the use of heuristics, which although they do not guarantee the exact solution, establish a compromise between the results obtained and the computational cost [3].*

We will present below some issues that were considered for the implementation of our algorithm such as the use of computer memory and the processing speed.

III. QUESTIONS CONSIDERED

The algorithm we will present here depends on having some permutations lists available. The necessary lists are these:

1. Permutation 3 → [1,2,3] : 6 combinations
2. Permutation 4 → [1,2,3,4] : 24
3. Permutation 5 → [1,2,3,4,5] : 120
4. Permutation 6 → [1,2,3,4,5,6] : 720
5. Permutation 7 → [1,2,3,4,5,6,7] : 5.040
6. Permutation 8 → [1,2,3,4,5,6,7,8] : 40.320
7. Permutation 9 → [1,2,3,4,5,6,7,8,9] : 362.880

The permutation lists with less than 3 elements are unnecessary since for a single city the route has size 0 and for two cities the route formed by path **A** → **B** → **A** is unique and therefore the shortest. The algorithm we use to generate these lists generates the combinations of 3, 4, 5, 6 and 7 elements by brute force. The 8 and 9 element lists are generated by joining adapted lists of 4 and 5 elements that contain 4 or 5 digits, respectively, of a set of 8 or 9 digits, respectively, per record. The lists were generated successfully in a timely manner.

Another issue considered was deciding whether the use of genetic algorithms would be useful for this problem. Given the way our algorithm was structured we decided not to use a pseudo-random genetic algorithm precisely to test the efficiency of our algorithm in a deterministic way. However, the group joining mechanism does a similar job as the genetic algorithm. Pacheco [7] talks about his experience with genetic algorithms applied to solve the problem of the traveling salesman:

> *"The use of genetic algorithms to solve the problem of the traveling delivery man was motivated by the fact that there is an algorithm that can solve the optimum problem for only a few vertices. For larger problems, there are only heuristics to obtain sub-optimal solutions. With the genetic algorithm it was expected to obtain better results than the heuristics (possibly optimal), which had a lower computational cost than the algorithm that obtains the optimal. (...) The big problem found was that the genetic algorithm did not find better results nor compared to the heuristics of the traditional algorithm, which is very fast". [7]*

Another important issue is processing time and memory usage. In this case we had no problems. The algorithm can calculate a route for 1,000 cities in just over 3 minutes, using about 2.3 GB of memory and 4 CPUs working in parallel[1].

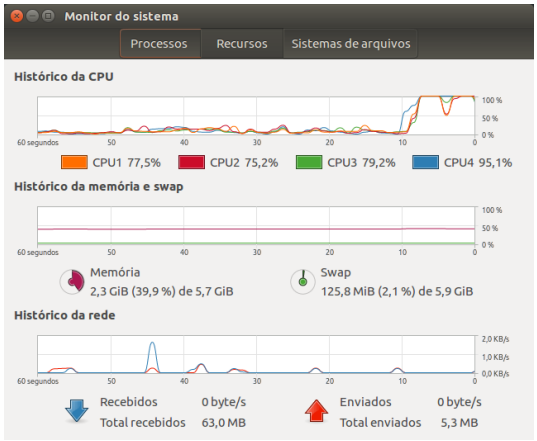

Figure 1: Memory map and CPU resources during the execution of the algorithm for a 1,000 city route.

An important note is that we decided to save the distances between cities in a matrix to increase the processing speed. This implies that our algorithm, as written, will have a limitation in the number of cities, for the purpose of calculating the shortest route, due to exhaustion of computer memory.
For 10,000 cities the memory used reached 5.2 GB and the parallel execution of the 4 cores of the Intel i5 processor reached 100% of resources for each of them, that is, the notebook was, so to speak, at the "limit of its forces".

1 A notebook with Intel® processor Core™ i5-3210M CPU @ 2.50GHz × 4 , 6 GB memory, Ubuntu 16.04 LTS 64-bit operating system was used.

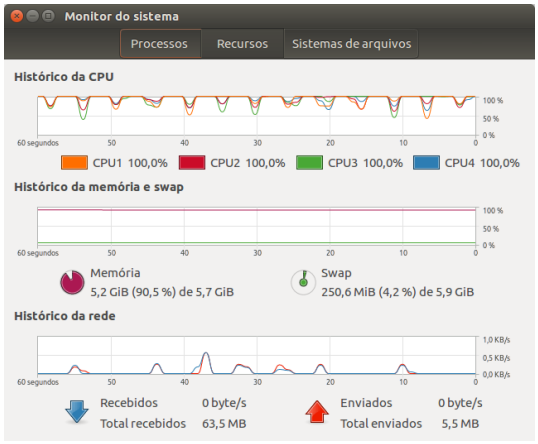

Figure 2: Memory map and CPU resources during the execution of the algorithm for a 10,000 city route.

This memory limitation can be easily remedied by eliminating the distance matrix between cities and checking the distance directly at each route calculation. The algorithm processing will be a little slower but it would be possible to increase the number of cities a little more as long as there are processor resources.

IV. FEATURING THE ALGORITHM

The algorithm we designed uses the principle of the optimum solution through brute force with the division of the problem to achieve a viable and fast solution when the number of cities is very large.

If the number of cities is less than or equal to 10 the problem is solved through the deterministic algorithm (testing all possible routes). As the maximum number of combinations for 10 cities is 9! (362.880) this is perfectly feasible.

If the number of cities is greater than 10 then we divide the cities into groups of 10 and calculate the optimal route for each of these groups. Before starting this process we should sort the cities, starting with **city 1** and then continuing with the others in the order of distance that **city n** has from **city 1** (starting point). This sorting is very important because it allows us to identify which are the cities closest to the starting point.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
For each of the 10 groups 362.880 routes are analyzed The starting point of group 1 is city 1 The starting point of group 2 is the city 11 The starting point of group 3 is the city 21 And so on									

Table 2: Shows the division of a set of 100 cities

The second part of the algorithm focuses on the junction points of each group. The algorithm calculates the additions or decreases in route value from the permutation of the last 4 and the first 4 elements of each group pair.

Group 1						**Group 2**					
5	6	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	15	16
		With these 8 elements the algorithm makes the 40.320 permutations so that one finds a smaller route than the one already found considering the first 20 cities.									

Table 3: Shows the joining of different groups

Considering the hypothetical table of 100 cities we have 9 pairs of sequential groups so that we have to analyze 9 × 40.320 permutations, or 362.880 routes.

The junction point of groups can be analyzed from 7 different points. This new scenario gives us the possibility to test more possibilities to join two groups of cities already optimized in an optimal route (considering each group separately).

Group 1										**Group 2**									
1	2	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	12	13	14	15	16	17	18	19	20
1	2	3	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	13	14	15	16	17	18	19	20
1	2	3	4	5	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	14	15	16	17	18	19	20
1	2	3	4	5	6	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	15	16	17	18	19	20
1	2	3	4	5	6	7	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	16	17	18	19	20
1	2	3	4	5	6	7	8	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	17	18	19	20
1	2	3	4	5	6	7	8	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	18	19	20

Table 4: Shows the joining of different groups at different points

To test all the possibilities of joining groups we have to travel 7 × 362.880 routes, which results in 2.540.160 combinations. If the number of cities chosen is not a multiple of 10 the last incomplete block has a special treatment obeying its limits.

The summary table below shows the computational cost of the algorithm to generate a route in a set of 100 cities:

	WORK	**TOTAL OF COMBINATIONS**
1	Process the 10 city groups separately.	3.628.800
2	Processing route junction points	2.540.160
	Total routes to test:	**6.168.960**

Table 5: Table of the computational effort to find a route for 100 cities

The total number of routes for 100 cities is 99!, something around 9,332621544 × 10^{155}, an absolutely unfeasible number of routes to search. With our algorithm we have reduced the search space to just over 6.16 million combinations which is perfectly possible to run using a single computer.

V. RESULTS

To generate data for the traveling clerk problem, pseudo-random points are selected over an area of 1,000,000 km2 (1000 × 1000). These points are natural numbers in the range of 1 to 1000 and for each city two coordinates are required.

The distance between cities is calculated by the Pythagorean theorem, where the hypotenuse reveals the

distance between the points considering a two-dimensional space.

The algorithm is deterministic for a set of up to 10 cities (ie, finds the optimal solution, the shortest route). For sets of more than 10 cities you do not have the shortest route but a route smaller than the initial. Next we have data of a route for 5 cities:

CITY LOCATION

City	Column	Line
C1	268	114
C2	313	138
C3	521	234
C4	45	382
C5	989	40

Table 1: Location of the 5 cities

DISTANCE BETWEEN CITIES

Num.	City A	City B	Distance
[1]	C01	C02	51.000
[2]	C01	C03	280.016
[3]	C01	C04	348.645
[4]	C01	C05	724.788
[5]	C02	C03	229.085
[6]	C02	C04	362.436
[7]	C02	C05	683.067
[8]	C03	C04	498.478
[9]	C03	C05	506.616
[10]	C04	C05	1004.042

Table 2: Distance between cities

ROUTES FOUND

```
Best route..: [C01][C04][C03][C05][C02]: 2087,805
Worst route...: [C01][C03][C02][C04][C05]: 2600.367
Difference...: 512,562
```

Table 3: Routes found for 5 cities

For 5 cities the route calculation time is minimal. For 10 cities the time is approximately 2 seconds. We remind you that the optimal solution is being found. Let's see in the tables below a case for 10 cities.

CITY LOCATION

City	Column	Line
C01	753	331
C02	859	90
C03	489	368
C04	497	141
C05	758	733
C06	780	754
C07	993	725
C08	289	75
C09	201	441
C10	162	340

Table 4: Location of the 10 cities

ROUTES FOUND

```
Best route..: [C01][C07][C08][C09][C10][C02][C05]
[C06][C03][C04]: 4355.347
Worst route.: [C01][C06][C07][C08][C10][C02][C03]
[C04][C05][C09]: 5160.023
Difference..: 804.676
```

Table 5: Routes found for 10 cities

These two examples show the optimal route for each case. However, for a larger number of cities, there is no way to obtain the optimal route due to the combinatorial explosion problem. In this case our algorithm selects a searchable set from all possible routes, of course, without evaluating all routes. The table below shows some results:

Number of cities	Route Initial	Route found	Comparison Reduction (%)	Algorithm run-time
25	11.304.941	6.369.950	56,34%	4 sec.
50	24.098.151	12.387.283	51,40%	9 sec.
75	40.279.086	16.772.864	41,64%	14 sec.
100	49.858.463	20.336.077	40,78%	18 sec.
150	79.005.689	25.984.490	32,88%	27 sec.
250	126.067.231	38.866.095	30,82%	47 sec.
500	271.146.655	70.504.488	26,00%	1 m 32 sec.
750	401.423.531	107.817.185	26,85%	2 m 19 sec.
1000	518.458.473	160.797.248	31,01%	3 m 5 sec.

Table 6: Shows information about the algorithm processing for multi-city scenarios.

Table 6 shows the processing results for sets ranging from 25 to 1000 cities. The processing time is very good (just over 3 minutes for 1000 cities) and the level of reduction of routes compared to the initial route has reached 26%.

Table 7 shows the results of tests applied to sets of 5000 or more cities.

Number of cities	Route Initial	Route found	Comparison Reduction (%)	Algorithm run-time
5.000	2.581.943.589	768.084.115	29,74%	18 min.
5.000	2.601.570.648	671.360.146	25,80%	15 m 31 sec.
7.500	3.912.821.962	1.027.090.615	26,24%	23 m 33 sec.
7.500	3.946.900.932	1.004.572.210	25,45%	23 m 32 sec.
7.777	4.052.261.825	1.147.234.520	28,31%	25 m 18 sec.
10.000	5.236.526.037	1.359.459.510	25,96%	32 m 45 sec.
10.000	5.218.839.793	1.650.957.590	31,63%	31 m 45 sec.
12.500	6.531.562.713	1.808.435.581	27,68%	40 m 5 sec.

Table 7: Shows information about the algorithm processing for sets with more than 5000 cities.

In the tests we observed that the route found for a set of cities above 150 is always close to 30% of the original (with variations for more and for less depending on the location of each city).

Another test we conducted was to compare the performance of our algorithm using the bier127 challenge that is available along with others at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.

This site contains several challenges related to the traveling clerk problem and other related issues. Actually TSPLIB [6] is a library of example instances for the TSP (Traveling salesman problem) and related problems from various sources and of various types. The bier127 challenge provides cities with the following coordinates:

City	X	Y	City	X	Y
1	9860	14152	**33**	8236	11020
2	9396	14616	**34**	8468	12876
3	11252	14848	**35**	8700	14036
4	11020	13456	**36**	8932	13688
5	9512	15776	**37**	9048	13804
6	10788	13804	**38**	8468	12296
7	10208	14384	**39**	8352	12644
8	11600	13456	**40**	8236	13572
9	11252	14036	**41**	9164	13340
10	10672	15080	**42**	8004	12760
11	11136	14152	**43**	8584	13108
12	9860	13108	**44**	7772	14732
13	10092	14964	**45**	7540	15080
14	9512	13340	**46**	7424	17516
15	10556	13688	**47**	8352	17052
16	9628	14036	**48**	7540	16820
17	10904	13108	**49**	7888	17168
18	11368	12644	**50**	9744	15196
19	11252	13340	**51**	9164	14964
20	10672	13340	**52**	9744	16240
21	11020	13108	**53**	7888	16936
22	11020	13340	**54**	8236	15428
23	11136	13572	**55**	9512	17400
24	11020	13688	**56**	9164	16008
25	8468	11136	**57**	8700	15312
26	8932	12064	**58**	11716	16008
27	9512	12412	**59**	12992	14964
28	7772	11020	**60**	12412	14964
29	8352	10672	**61**	12296	15312
30	9164	12876	**62**	12528	15196
31	9744	12528	**63**	15312	6612
32	8352	10324	**64**	11716	16124

Table 8: first 64 cities of the bier127 challenge

City	X	Y	City	X	Y
65	11600	19720	**97**	5336	10324
66	10324	17516	**98**	812	6264
67	12412	13340	**99**	14384	20184
68	12876	12180	**100**	11252	15776
69	13688	10904	**101**	9744	3132
70	13688	11716	**102**	10904	3480
71	13688	12528	**103**	7308	14848
72	11484	13224	**104**	16472	16472
73	12296	12760	**105**	10440	14036
74	12064	12528	**106**	10672	13804
75	12644	10556	**107**	1160	18560
76	11832	11252	**108**	10788	13572
77	11368	12296	**109**	15660	11368
78	11136	11020	**110**	15544	12760
79	10556	11948	**111**	5336	18908
80	10324	11716	**112**	6264	19140
81	11484	9512	**113**	11832	17516
82	11484	7540	**114**	10672	14152
83	11020	7424	**115**	10208	15196
84	11484	9744	**116**	12180	14848
85	16936	12180	**117**	11020	10208
86	17052	12064	**118**	7656	17052
87	16936	11832	**119**	16240	8352
88	17052	11600	**120**	10440	14732
89	13804	18792	**121**	9164	15544
90	12064	14964	**122**	8004	11020
91	12180	15544	**123**	5684	11948
92	14152	18908	**124**	9512	16472
93	5104	14616	**125**	13688	17516
94	6496	17168	**126**	11484	8468
95	5684	13224	**127**	3248	14152
96	15660	10788			

Table 9: last 65 cities of bier127 challenge

With this data we tested our algorithm which returned the following result in 23 seconds:

[C001][C006][C007][C008][C010][C002][C003][C004][C005]
[C018][C016][C020][C009][C012][C013][C019][C028][C026]
[C029][C021][C015][C011][C014][C017][C030][C022][C023]
[C024][C025][C027][C032][C033][C037][C040][C038][C036]
[C031][C034][C035][C048][C042][C043][C050][C039][C047]
[C056][C045][C046][C051][C044][C049][C041][C057][C058]
[C060][C052][C053][C054][C055][C063][C061][C070][C059]
[C066][C067][C068][C062][C064][C071][C072][C065][C069]
[C079][C080][C077][C078][C073][C088][C076][C089][C090]
[C087][C075][C081][C074][C082][C085][C091][C099][C098]
[C086][C084][C100][C097][C083][C092][C095][C096][C093]
[C094][C108][C103][C107][C109][C102][C101][C110][C104]
[C118][C117][C116][C120][C111][C112][C106][C115][C114]
[C105][C113][C119][C121][C122][C123][C125][C126][C127]
[C124]

This route has a size of 247.163.183 and the initial route has a size of 393.998.276. The route found represents 62.73% of the original. A much smaller reduction than we observed in other tests. The best known solution to the bier127 challenge has a route of 118,282 that represents 30.02% of the original route size. The optimal solution, however, begins with the city number twelve[2].

12 - 14 - 41 - 36 - 37 - 35 - 40 - 43 - 34 - 42 - 39 - 38 - 26 – 25 – 33
122 - 28 - 29 - 32 - 98 - 97 - 123 - 95 - 93 - 127 - 107 – 111 – 112
94 - 46 - 118 - 48 - 53 - 49 - 47 - 55 - 66 - 113 - 65 - 99 – 92 – 89
125 - 104 - 110 - 85 - 86 - 87 - 88 - 109 - 96 - 119 - 63 – 102 – 101
83 - 82 - 126 - 81 - 84 - 117 - 78 - 76 - 75 - 69 - 70 - 71 – 68 – 74
73 - 67 - 8 - 72 - 19 - 22 - 4 - 23 - 24 - 9 - 11 - 3 - 90 – 116 – 60
59 - 62 - 61 - 91–58 - 64 - 100 - 10 - 120 - 13 - 115 - 50 – 5 – 52
124 - 56 - 121 - 57 - 54 - 45 - 103 - 44 - 51 - 2 - 16 - 1 – 7 – 105
114 - 6 - 106 - 15 - 108 - 20 - 17 - 21 - 18 - 77 - 79 - 80 – 31 – 27
30

Our algorithm is not sophisticated and for this reason it is not able to reach levels of route reduction comparable to the best algorithms that are used to overcome these challenges.

VI. CONCLUSION

The problem of the traveling salesman has been studied for a long time and there is still no algorithm to find the optimal route in time (polynomial). Our goal with this work was to learn a little about the problem and test an algorithm that we built to find good routes between cities. Our goal was partially achieved since the analyzed algorithm found better routes than the initial one in a very short time, mainly for sets of up to 1000 cities. Improving this performance is a complex job. About this complexity Porto da Silveira says:

> *"Don't be fooled by the playful appearance of this problem. It is NOT another inconsequential curiosity to entertain unmotivated students. In fact, it is a problem that should be part of the baggage of every competent math professional". [8]*

We then concluded that our algorithm, although simple, proved effective in finding better than initial routes between a set of cities. We tested sets of up to 12.500 cities and had route reduction levels of between 26% and 62% compared to the initial route.

2 Available at: https://github.com/ashleywang1/TSP-solver/blob/master/TSP/2-opt-p/2optpresult/bier127.tour

REFERENCES

[1] Xuesong Yan, Can Zhang, Wenjing Luo, Wei Li, Wei Chen, Hanmin Liu. **Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm.** Wuhan, Hubei 430050, China. Disponível em: <http://www.ijcsi.org/papers/IJCSI-9-6-2-264-271.pdf >. Acessado em: 24/05/2017, 12:30h

[2] Vignatti, André. **P, NP e NP-Completo**. DINF- UFPR. Disponível em: <http://www.inf.ufpr.br/vignatti/courses/ci165/23.pdf> Acessado em: 24/05/2017, 12:43h.

[3] Colares, Flávio Martins; Silva, José Lassance de Castro; Silva, José Ramos de Oliveira; Carvalho, Maria do Socorro de. **Uma heuristica aplicada ao problema do caixeiro viajante.** XXXVII simpósio brasileiro de pesquisa operacional. 27 a 30/09/05, Gramados. Disponível em: <http://www.repositorio.ufc.br/bitstream/riufc/13121/1/2005_eve_jclsilva.pdf>. Acessado em: 24/05/2017, 13:00h.

[4] Araujo, Silvio Alexandre de. **Heurísticas para Otimização Combinatória.** UNESP. Disponível em: <https://www.dcce.ibilce.unesp.br/~saraujo/disciplinas/Metaheuristicas.pdf>. Acessado em 24/05/2017, 15:11h.

[5] Wikipedia. **Universo observável.** Disponível em:<https://pt.wikipedia.org/wiki/Universo_observ%C3%A1vel>, Acessado em 25/05/2017.

[6] Reinelt, Gerhard. **TSPLIB**. TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types. Disponível em: <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html> Acessado em: 25/05/2017, 13:10h.

[7] Pacheco, Marco Aurélio; Fukasawa, Ricardo. **Resolução do problema do entregador viajante**. Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro. Disponível em: <http://rica.ele.puc-rio.br/media/Revista_rica_n4_a8.pdf> Acessado em: 26/05/2017.

[8] Silveira, J.F. Porto da. **Problema do caixeiro viajante.** UFRGS, 29-junho-2000. Disponível em: <http://www.mat.ufrgs.br/~portosil/caixeiro.html> Acessado em: 28/05/2017, 00:00h