Hard and Easy to Solve TSP Instances

María A. Osorio, David Pinto
School of Computer Sciences,
Universidad Autónoma de Puebla,
Ciudad Universitaria, Puebla 72560, México
{aosorio, dpinto}@cs.buap.mx}

Abstract

The main purpose of this work is to bring together ideas from Artificial Intelligence (AI) and Operation Research (OR) fields and apply different distributions to generate hard instances for TSP. We solved the problems generated with Branch and Bound, and used the number of nodes in the searching tree, needed to solve the problems to optimality, as a measure of the hardness of the instances. These instances were generated with a random uniform distribution, and an exponential random distribution.

1. Introduction

A critical point in algorithm development issue usually arises with the need of comparing their performance. The set of benchmark problems is often comparatively small, and it is hard to be sure that a good performance is not the result of good luck. Formal theorems on performance are solid, but are often hard to come by and may not be relevant to practical problems. Algorithms can, of course, be tested on random problems, but there is justifiable suspicion of such results, as random problems are not meaningful in themselves. This work can be seen as a contribution in the sense of the "empirical science of algorithms" [14].

The idea of generating specially hard instances to test algorithms have been studied by the AI community since 1987, with the studies of Huberman[11] on phase transitions applied to coloring and satisfiability problems. More recent studies in phase transitions applied to satisfiability problems can be seen in Kirkpatrick *et al* [21] and Mitchell *et al* [22]. However, these authors suggested that it would be impossible to find a phase transition for TSP.

NP-complete problems can be summarized by at least one "order parameter", and, usually, hard problems occur at a critical value of such a parameter [1]. This critical value separates two regions for characteristically different properties. For example, for coloring or satisfiability problems, there is a critical value that separates overconstrained from underconstrained random graphs, and it marks the value at which the probability of a solution changes abruptly from near 0 to near 1. It is the high density of well-separated almost solutions (local minima) at this boundary that cause search algorithms, as branch and bound to "thrash". This boundary is known as the phase transition of a NP-complete problem [11].

To date most of the research has been directed at studies of k-SAT, graph coloring and the traveling salesman problem. In these cases the performance measure of interest has been some measure of computational search cost [6], [7], [8].

In a paper with the title "Where the Really Hard Problems are?", Cheeseman et al [1] initiate the adventure of looking for a phase transition in TSP. They extrapole regions from Hamiltonian Circuits problems, and present the first approximation to a phase transition for TSP, using a log-normal distribution with a given standard deviation to generate the distance matrix between the cities. A decade later, the results obtained by Gent et al [6] and the implications of some of their papers [7], [8], showed that TSP can have a phase transition behavior. He compares the logarithm of the number of nodes in the search tree used for Branch and Bound and $l/(nA)$, where l is the total length of the tour, n the number of cities and A the area that cover all the cities. He found an approximating phase transition when $l/\sqrt{(nA)} \cong 0.6$.

While phase transition concepts have got a lot of attention from the AI community, must people in the OR community seems to ignore these results and continue testing algorithms in random generated instances. However, the concept of using the number of nodes in a searching tree as a measure of hardness, has the origin in mathematical programming concepts and relies on the integer programming formulation of the problem. For this reason, it has already been used with the same purpose by people from the OR community (see Hooker et al[15] and Osorio et al [25]).

The main purpose of this work is to bring together ideas from both communities and apply the concepts of distribution used by the OR community to generate hard instances for TSP.

To relate the experience obtained in this research, we structured the present paper in the following way. In section 2, we introduce the TSP problem, and present the

Integer Programming formulation of TSP in section 3. In section 4, we describe Easy and Hard generators of distributions for the distance matrix in TSP. Section 4 shows computational results and section 5, the conclusion.

2. Traveling Salesman Problem (TSP)

The TSP has received great attention from the operations research and computer science communities because is very easy to describe but very hard to solve [5]. The problem can be formulated saying that the traveling salesman must visit every city in his territory exactly once and then return to the starting point. Given the cost of travel between all cities, he should plan his itinerary for a minimum total cost of the entire tour.

Space solution for TSP is the n-cities permutation, $n!$. Any simple permutation is a different solution. The optimum is the permutation that correspond to a travel with the minimum cost. The evaluation function is very simple, because we only need to add the cost profit associated with each segment in the itinerary, to obtain the total cost for that itinerary [2].

The TSP is a relatively old problem. It was already documented in 1759, with a different name, by Euler. The term 'traveling salesman' was first used in 1932 in a German book written by a veteran traveling salesman. The TSP, in the way we know it now, was introduced by the RAND Corporation in 1948. The Corporation's reputation helped to make the TSP a well known and popular problem. The TSP also became popular at that time due to the apparition of linear programming and the attempts to solve combinatorial problems.

In 1979, it was probed that the TSP is NP-hard, a special kind of NP-complete problems (see Garey *et al*, [4]). All NP problems can be polynomially reduced to them. It means that if one can find a solution in polynomial time to one of them, with a deterministic procedure, it may find it for all NP and then, P=NP. Nobody has been able to find efficient algorithms for NP-complete problems until now, and nobody has demonstrated that such algorithms do not exist.

The TSP can be symmetric or asymmetric. In the symmetric case, departure and return costs are the same and can be represented with an undirected graph. For the asymmetric case, the more common one, the departure and return costs are different and can only be represented by a directed graph. Because the symmetric problem can be seen as a special case of the asymmetric one, we present the integer programming formulation for the asymmetric case in the next section.

On the other way, under the same settings, symmetric TSP instances need considerable more nodes in the searching tree and CPU time to reach an optimal solution. For that reason, the computational experiments in section 5 were applied to the symmetric case [17].

The TSP has become a classic problem because it serves to represent a great number of applications in real life, as the coloring sequence in textile industry, the design of insulating material and optic filters, the impression of electronic circuits, the planning of trajectories in robotics and many other examples that can be represented using sequences (see Salkin [27]). Besides, it may represent a big number of combinatorial problems that cannot be solved in polynomial time and are NP hard.

The exponential nature of the time needed to solve this problem in an exact way has originated, during the last decades, a big amount of research directed to find exact algorithms for specific cases and the development of heuristic algorithms to approximate its optimal solution (see [5], [26]). Besides, it has been widely used by the AI and OR communities to establish benchmarks in algorithms.

3. Integer Programming Formulation

As we mentioned before, a traveling salesman must visit n cities, each exactly once. The distance between every pair of cities ij, denoted by d_{ij} ($i \neq j$), is known and may depend on the direction traveled (*i.e.*, d_{ij} does not necessarily equal d_{ji}). The problem is to find a tour which commences and terminates at the salesman's home city and minimizes the total distance traveled.

Suppose we label the home city as city 0 and as city $n+1$. (Then we may think of the salesman's initial location as city 0 and the desired final location as city $n+1$). Also, introduce the zero-one variables x_{ij} (i=0,1,…,n, j=1,…,$n+1$, $i \neq j$), where $x_{ij} = 1$ if the salesman travels from city i to j, and $x_{ij} = 0$ otherwise. To guarantee that each city (except city 0) is entered exactly once, we have $\sum_{i=0,n} x_{ij} = 1$ ($j=1,...,n+1$, $i \neq j$).

Similarly, to ensure that each city (except city $n + 1$) is left exactly once, we have $\sum_{j=1,n+1} x_{ij} = 1$ ($i = 0,...,n$, $i \neq j$). These constraints, however, do not eliminate the possibility of subtours or "loops". One way of eliminating the subtour possibility is to add the constraints $\alpha_i - \alpha_j + (n+1)\, x_{ij} \leq n$ ($i = 0,...,n$, $j=1,...,n+1$, $i \neq j$).

Where α_i is a real number associated with city i. To complete the model we should minimize the total distance between the cities. An integer programming formulation of the traveling salesman problem is to find variables x_{ij} and arbitrary real numbers α_i which

$$\text{Minimize} \qquad \sum_{i=0,n} \sum_{j=1,n+1} d_{ij} x_{ij}$$

$$\text{Subject to} \quad \sum_{i=0,n} x_{ij} = 1 \qquad (j=1,\ldots,n+1,\ i \neq j)$$

$$\sum_{j=1,n+1} x_{ij} = 1 \qquad (i = 0,\ldots,n,\ i \neq j)$$

$$\alpha_i - \alpha_j + (n+1)\, x_{ij} \leq n \qquad (i = 0,\ldots,n,\ j=1,\ldots,n+1,\ i \neq j)$$
$$\alpha_i \geq 0 \qquad (i = 0,\ldots,n+1)$$
$$x_{ij} \in \{0,1\}, \qquad (i = 0,\ldots,n,\ j=1,\ldots,n+1,\ i \neq j)$$

This formulation was first presented by Tucker [28]. We illustrate the integer programming formulation in the following example. Table 1 shows the distances for a traveling salesman problem with 3 cities.

Table 1. Distances for the 3-cities example

From/To	1	2	3
1	∞	26	82
2	134	∞	117
3	38	13	∞

The Integer Programming formulation for this example is:

$$\text{Minimize } 26\, x_{12} + 82\, x_{13} + 134\, x_{21} + 117\, x_{23} + 38\, x_{31} + 13\, x_{32}$$
$$\text{Subject to} \quad x_{01} + x_{02} + x_{03} + x_{04} = 1$$
$$x_{12} + x_{13} + x_{14} = 1$$
$$x_{21} + x_{23} + x_{24} = 1$$
$$x_{31} + x_{32} + x_{34} = 1$$
$$x_{01} + x_{21} + x_{31} = 1$$
$$x_{02} + x_{12} + x_{32} = 1$$
$$x_{03} + x_{13} + x_{23} = 1$$
$$x_{04} + x_{14} + x_{24} + x_{34} = 1$$
$$4x_{01} + \alpha_0 - \alpha_1 \leq 3$$
$$4x_{02} + \alpha_0 - \alpha_2 \leq 3$$
$$4x_{03} + \alpha_0 - \alpha_3 \leq 3$$
$$4x_{04} + \alpha_0 - \alpha_4 \leq 3$$
$$4x_{12} + \alpha_1 - \alpha_2 \leq 3$$
$$4x_{13} + \alpha_1 - \alpha_3 \leq 3$$
$$4x_{14} + \alpha_1 - \alpha_4 \leq 3$$
$$4x_{21} + \alpha_2 - \alpha_1 \leq 3$$
$$4x_{23} + \alpha_2 - \alpha_3 \leq 3$$
$$4x_{24} + \alpha_2 - \alpha_4 \leq 3$$
$$4x_{31} + \alpha_3 - \alpha_1 \leq 3$$
$$4x_{32} + \alpha_3 - \alpha_2 \leq 3$$
$$4x_{34} + \alpha_3 - \alpha_4 \leq 3$$
$$\alpha_i \geq 0 \qquad (i = 0,\ldots,4)$$
$$x_{ij} \in \{0,1\}, \qquad (i = 0,\ldots,3,\ j=1,\ldots,4,\ i \neq j)$$

4. Hard and Easy Problems Generators

Problem generation methodologies are currently a very important topic of research. A main part of applying new techniques consists of empirically testing algorithm performance across a representative range of problems [14]. An inadequate set of test problems can provide misleading information about algorithm performance.

Laguna, *et al.* [21] developed a procedure to create hard problem instances for MGAP (Multilevel Generalized Assignment Problems). This approach embodies a random problem generator, labeled E, that draws constraint coefficients from an exponential distribution and generates the coefficients in the objective function to be inversely correlated. The effectiveness of this approach for generating difficult problems led Osorio *et al.* [24] to consider adapting some of its features to the MKP (Multidimensional Knapsack Problems) setting.

Combining all the ideas described above for generating hard problems, Osorio *et al.* [25] developed a generator that produces challenging MKP problems. Their approach uses independently exponential distributions over a wide range to generate the constraint coefficients, and the corresponding average for each variable is used to calculate directly correlated coefficients in the objective function. The RHS values are set to 0.25 of the sum of the corresponding constraint coefficients for the first part of the experiment, using the concept that tighter constraints yield difficult problems (see [3], [9]). In the second part of the experiment, this multiple was set at 0.25, 0.5 and 0.75.

Inspired by the results obtained for MKP problems, we used exponential distribution to generate the traveling distance between the cities, where d_{ij} are integer numbers drawn from the exponential distribution: d_{ij}=1.0–1000 $ln(U(0,1))$, $(i = 0,\ldots,n,\ j=1,\ldots,n+1,\ i \neq j)$.

We compared the number of nodes with the results obtained when we generate the traveling distance between the cities, where d_{ij} are integer numbers drawn from uniform distribution: $d_{ij}=U(0,1)$, $(i = 0,\ldots,n,\ j=1,\ldots,n+1,\ i \neq j)$.

The measure of the "hardness" of an integer problem has been studied by Hooker *et al* [13],[15],[16]. He concludes that the search tree size, expressed by the number of nodes, is relatively an intrinsic measure. It does not depend on most of the details of the software. The main problem is that it depends on the branching rule, but we can solve problems using the same branching rule and platform. It may be, sometimes, done for two or three branching rules to see if it affects the relative results.

5. Computational Results

All problems were solved on a Pentium III, 1066 MHz and 248 MB RAM Memory, using the general purpose software for optimization, CPLEX V7.5. This software uses the branch and bound method to solve integer

programming problems. The branch and bound method is a complete exhaustive technique that utilizes a searching tree to obtain the optimal integer solution. The number of nodes in the searching tree can be used as an intrinsic measure of the hardness of an integer program (see Hooker[15]).

We tested our ideas in 40 instances generated with a random uniform distribution, and 40 instances generated with an exponential random distribution and compared the number of nodes and the seconds of CPU needed to reach optimality in the searching tree. As we explained in the first section, the computational experiments were applied to the symmetric case and both set of problems were generated with the equations described in the last section.

The first column shows the number of cities and the next two, the number of nodes in the searching tree and the CPU time needed to solve the problems generated with the uniform distribution to optimality. Column 4 shows the optimal value for the instances. The next three columns indicate the same data for the instances generated with the exponential distribution.

	Uniform Distribution			Exponential Distribution		
Cities	Nodes	CPU	Optimum	Nodes	CPU	Optimum
10	4	0.26	1698	0	0.04	1599
10	225	0.27	859	0	0.03	1882
10	88	0.2	1136	18	0.11	1465
10	20	0.09	1305	3	0.06	1723
10	123	0.21	1294	115	0.2	2011
AVG	**92**	**0.206**	**1258.4**	**27.2**	**0.088**	**1739.25**
20	10	0.46	1386	30	0.47	1989
20	0	0.15	1494	94	0.85	2542
20	0	0.11	1479	95500	162.57	1615
20	24827	49.74	1374	0	0.12	2107
20	123	1.06	1677	7678	14.86	2504
AVG	**4992**	**10.304**	**1482**	**20660.4**	**35.774**	**2151.4**
30	49	1.89	1681	482	11.13	1601
30	382	9.07	1597	363	5.1	1926
30	6	0.98	1716	1049	12.58	1844
30	1391	16.54	1911	5963	39.92	1240
30	51	1.51	1413	1049	13.8	1844
AVG	**375.8**	**5.998**	**1663.6**	**1781.2**	**16.506**	**1691**
40	1	3.38	1922	1957	60.06	2116
40	0	0.79	1827	41163	392.49	1556
40	0	0.59	1574	7366	120.26	2044
40	8813	118.84	1212	33763	382.25	1855
40	4339	91.25	1741	55397	12835	2170
AVG	**2630.6**	**42.97**	**1655.2**	**27929.2**	**2758.01**	**1948.2**

Table 2 Nodes, CPU cycles & Optimum comparison.

6. Conclusions

The results confirm that the problems generated with the exponential distribution need a greater number of nodes (in average) in the searching tree and CPU time to be solved to optimality for all the instances. These results highlight the need to combine efforts from the AI and OR fields to explore more in detail the behavior of combinatorial problems to get more information about the settings that make “hard” problems really hard.

References

[1] P. Cheeseman, B. Kanefsky and W. M. Taylor, "Where the Really Hard Problems Are", in *Proceedings of the 12th IJCAI, International Joint Conference on Artificial Intelligence*, (1991) 331-337.

[2] G.B. Dantzig, Discrete Variables Problems, Operations Research 5 (1957) 266-277.

[3] A. Fréville and G. Plateau, "Hard 0-1 Multiknapsack Test Problems for Size Reduction Methods", Investigación Operativa **1** (1990), 251-270.

[4] M. Garey and D. Johnson, *Computers and Intractability*, W.H. Freeman, San Francisco (1979).

[5] S. Gass (ed.), *Encyclopedy of Operations Research and Management Sciences*, Kluwer Academic Publishers, New York (1997).

[6] I. P. Gent, and T. Walsh, "An empirical analysis of search in GSAT", Journal of Artificial Intelligence Research, **1** (1993) 47-59.

[7] I. P. Gent, and T. Walsh (1994), "Easy problems are sometimes hard", Artificial Intelligence, **70** (1994) 335-345.

[8] I. P. Gent, and T. Walsh (1996), "The TSP phase transition", Artificial Intelligence, **1-2** (1996) 349-358.

[9] R. Hill and Ch. Reilly, "The Effects of coefficient correlation Structure in Two-Dimensional Knapsack Problems on solution Procedure Performance", Management Science **46** (2000) 302-317.

[10] T. Hogg and C. Williams, "The hardest constraint problems: A double phase transition", Artificial Intelligence, **69** (1994) 359-377.

[11] T. Hogg, B. A. Huberman, and C. Williams, "Phase Transitions and the Search Problem", Artificial Intelligence, **81, 1-2** (1996) 1-15.

[12] T. Hogg, "Refining the Phase Transition in Combinatorial Search", Artificial Intelligence, **81, 1-2** (1996) 127-154.

[13] J. N. Hooker, "Logic-based methods for optimization", in A. Borning, ed., Principles and Practice of Constraint Programming, Lecture Notes in Computer Science **874** (1994) 336-349.

[14] J. N. Hooker, "Needed: An Empirical Science of Algorithms". Operations Research **42** (1994) 201-212.

[15] J. N. Hooker and M. A. Osorio, "Mixed Logical/Linear Programming". Discrete Applied Mathematics **96-97** (1999) 395-442.

[16] J. N. Hooker, "A Framework for combining solution methods". Working Paper, Carnegie Mellon University (2003).

[17] R. E. Jeroslow and J. K. Lowe, "Modeling with integer variables", Mathematical Programming Studies **22** (1984), 167-184.

[18] D. S. Johnson, "Local Optimization and the Traveling Salesman Problem", in Proceedings of the 17th International Colloquium on Automata, Languagues and Programming, (1990) 446-461, Springer, Berlin.

[19] D. S. Johnson, L. A., "The Traveling Salesman Problem: A case Study in Local Optimization. In: E.H.L. Aarts, J. K. Lenstra (eds.): *Local Search in Combinatorial Optimization*. John Wiley and Sons, Ltd (1997) 215-310.

[20] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovich, "Experimental Analysis of Heuristics for the ATSP", In: G. Gutin, G. and A. Punnen, A. (eds*.): The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers, Dordrecht (2002) 445-487.

[21] S. Kirkpatrick and B. Selman, "Critical behavior in the satisfiability of random boolean expressions", Science, (1994).

[22] D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of SAT problems". In: Proceedings, 10th National Conference on Artificial Intelligence. AAAI Press/The MIT Press (1992).

[23] M. Laguna, J. P. Kelly, J.L, González-Velarde, F., Glover, "Tabu search for the multilevel generalized assignment problem", European Journal of Operational Research **82** (1995) 176-189.

[24] M. A. Osorio, and F. Glover, "Hard Problem Generation for MKP". Proceedings of the XI CLAIO, Concepción, Chile (2002).

[25] M. A. Osorio, F. Glover, and P. Hammer, "Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions". *Annals of Operations Research* **117** (2002), 71-93.

[26] M. A. Osorio, and D. Pinto, "A Preprocessing that Combines Heuristic and Surrogate Constraint Analysis to Fix Variables in TSP". Submitted to MICAI 2004, Springer Verlag.

[27] M. Salkin, M, *Integer Programming*. Adisson-Wesley Publishing Company. New York (1975)

[28] A. Tucker, "On Directed Graphs and Integer Programs". IBM Mathematical Research Project Technical Report, Princeton University (1960)