INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No. 667/ November 2010

Cross Entropy based Neighborhood Reduction and Initial Tour Formation for Tabu Search on the Asymmetric Traveling Salesman Problem

by

Sumanta Basu
Assistant Professor, IIM Calcutta, Diamond Harbour Road, Joka P.O., Kolkata 700 104 India

&

Diptesh Ghosh
Associate Professor, IIM Ahmedabad, Vastrapur, Ahmedabad Pin-380015, India

Cross Entropy based Neighborhood Reduction and Initial Tour Formation for Tabu Search on the Asymmetric Traveling Salesman Problem

Sumanta Basu* Diptesh Ghosh †

Abstract

The objective of this paper is to implement tabu search on moderate sized asymmetric traveling salesman problems (ATSPs). We introduce a preprocessing scheme based on the cross entropy method which allows us to reduce the number of arcs in the graph defining an ATSP instance, without significantly affecting the cost of the tour output by tabu search. This reduction helps us to apply tabu search methods especially designed for ATSPs defined on sparse graphs. We also provide a scheme to generate good initial tours for multi-start tabu search to run on large problems. We report our computational experiences on randomly generated problems as well as benchmark problems to show that our method yields good quality tours for moderate sized ATSPs much faster than conventional tabu search implementations.

1 Introduction

Consider a digraph $G = (V, A)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of n nodes, and $A = \{(v_i, v_j)|v_i, v_j \in V\}$ is a set of arcs. Each arc (v_i, v_j) has a cost c_{ij}. Its density is defined as $\rho = |A|/\{|V|(|V| - 1)\}$. If $\rho = 1$, the digraph is called complete, and if ρ is significantly less than 1, the graph is called sparse. If the existence of an arc (v_i, v_j) in A implies that (a) $(v_j, v_i) \in A$, and (b) $c_{ij} = c_{ji}$, then the graph is called symmetric, otherwise it is called asymmetric.

A tour in G is a simple directed cycle covering all nodes in V. The cost of a tour is the sum of costs of all arcs in the tour. The traveling salesman problem (TSP) is one of finding a minimum cost tour in G. The cardinality n of V is called the size of the TSP. If a TSP is defined on a symmetric digraph, it is called a symmetric traveling salesman problem (STSP), otherwise it is

*OM Area, Indian Institute of Management Calcutta. Email: sumanta@iimcal.ac.in

†P&QM Area, Indian Institute of Management Ahmedabad. Email: diptesh@iimahd.ernet.in

called an asymmetric traveling salesman problem (ATSP). In this paper, we deal with ATSPs.

Karp (see Karp, 1972) showed that the Hamiltonian Cycle problem is $\mathscr{NP}$ complete, which implies that the TSP is $\mathscr{NP}$ hard, and so both exact algorithms (for an overview, see e.g., Applegate et al., 2006; Fischetti et al., 2002) and heuristic algorithms (for an overview, see e.g., Johnson et al., 2002) have been used in the literature to solve these problems.

In practice, for example in logistics, ATSPs occur more commonly than STSPs. However, most of the research in TSP has focused on the STSP. For STSPs, researchers have been successful in obtaining results which allow us to solve most large instances in very little time. However, such results are not available for the ATSP. According to Johnson et al. (2002), one reason for this could be the absence of particular instance type for the ATSP which will enable algorithms to reduce computational time by exploiting specific problem characteristics. One line of approach (see, e.g., Cirassela et al., 2001) attempts to convert an ATSP instance into an equivalent STSP instance to obtain solutions. This approach has limited success since the size of the equivalent STSP instance is typically much larger than the size of the original ATSP instance. Implementation of metaheuristics on ATSP instances also has limited evidence of success in published literature. Basu and Ghosh (2008) notes that most of the literature on tabu search for the TSP is restricted to the STSP.

In this paper we develop tabu search implementations for the ATSP. We assume that the ATSP has been defined on a complete graph. Our aim is to make use of a tabu search implementation developed in Basu et al. (2008) which speeds up tabu search significantly in ATSP instances defined on sparse graphs. To that end, in Section 2, we devise a scheme to reduce the density of the graph underlying a TSP problem instance. This reduction may eliminate an optimal tour, but that fact is not of grave concern to us, since we plan to use a tabu search algorithm on the sparse graph which is itself not guaranteed to output an optimal solution. It is interesting to note that there is almost no literature on preprocessing of TSP instances. On the contrary, preprocessing algorithms exist for other well-studied combinatorial problems, see for example, Goossens and Baruah (2001) for uniprocessor scheduling, and Khumawala (1975) for the uncapacitated facility location problem. Toth and Vigo (1998) and Glover and Laguna (1998) describe approaches that can be used for preprocessing, however none of the papers explicitly describes a preprocessing procedure.

Tabu search, when designed to solve large problems is often implemented in a multi-start manner, where it is run from multiple starting points suitably separated in the solution domain. The best solution encountered by tabu search from among all the runs is output as the final solution. In designing a multi-start tabu search implementation, one needs to ensure that the starting solutions are widely dispersed in the solution domain. In Sec-

tion 3, we describe ways in which the preprocessing method described in Section 2 can be modified to generate suitable initial tours for multi-start implementations of tabu search.

In Section 4 we describe our computational experience with randomly generated instances as well as benchmark problem instances. The random instances varied in size from 200 to 600. The benchmark instances were obtained from the collection of ATSP instances from Johnson et al. (2002). We chose only those instances in the collection which are defined on graphs with at least 100 nodes. We created three implementations, named A through C, such that comparisons among the implementations provide us with information about the efficiency of our preprocessing and initial tour generating operations. Finally, in Section 5 we conclude the paper with a summary of the contributions.

2 The Preprocessing Scheme

Consider an ATSP instance defined on a complete graph. Within its set of arcs, some are either too costly or too inconveniently located to be included in any low cost tour. Preprocessing is a process by which such arcs are eliminated from the graph. The sparse graph thus formed can then be addressed efficiently using tabu search implementations designed to work on sparse graphs (i.e., Basu et al., 2008), which make use of the fact that the neighborhood of a tour in a sparse graph is smaller than the neighborhood of the same tour in a complete graph. In the remainder of this section, we describe a preprocessing algorithm for this purpose. This algorithm is based on cross entropy. Cross entropy (CE) was developed as a tool for rare event simulation in Rubinstein (1997). This tool was later used in Rubinstein (1999, 2001) to solve combinatorial optimization problems. CE has been used for the TSP in Chepuri and Homem-de Mello (2005); Boer et al. (2005). It is an iterative process in which arcs that are less likely to be in good quality tours are progressively eliminated until only those arcs that are in the "best" tour remain. We use a truncated CE algorithm to reduce the number of arcs in a graph defining a TSP instance.

In our preprocessing algorithm, we take a directed graph $G = (V, A)$ as input, along with three parameters k, e, and $iter$. We define a probability matrix $P = [p_{ij}]$ where p_{ij} denotes the probability that arc (v_i, v_j) will be included in a random tour during a particular iteration. At the beginning of the preprocessing algorithm, $p_{ij} = 1/n(n-1)$ and $p_{ii} = 0$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$. The P matrix gets updated after each iteration of the algorithm. A typical iteration starts with a probability matrix P. During the iteration, we generate k tours in G such that the probability of an arc (v_i, v_j) being chosen in a tour is proportional to p_{ij}. We then create an elite tour list $\mathscr{E}$ containing the e lowest cost tours from among the k tours

generated. At the end of the iteration, we update P as follows. Let n_{ij} be the number of times that arc (v_i, v_j) appears in the tours in $\mathscr{E}$. Then $p_{ij} = n_{ij}/(e.n)$. After *iter* iterations get over, a sparse graph $G' = (V, A')$ is formed where $A' = \{(v_i, v_j) : p_{ij} > 0\}$.

From preliminary experiments we observed that at the end of the specified number of iterations, the output graph became too sparse and as a result some of the tours did not have any neighboring tour. In such graphs, tabu search was unable to better the best tour in the $\mathscr{E}$ list after *iter* iterations. So motivated by Toth and Vigo (1998), we added a step in our algorithm in which we chose a threshold τ, and added those arcs in A whose costs were less than the threshold to A'. The result of this operation is a denser G' but one in which some neighboring tours do exist for most tours.

Based on preliminary experiments with randomly generated ATSP instances of sizes 100 and 250, we chose k, e, and *iter* as 50000, $1.5n$ and 20 respectively. We chose the threshold value τ as 1.5 times the average of the costs of arcs in $\mathscr{E}$ weighed by the frequency of their appearance.

3 Generation of Initial Tours

Performance of any local search heuristic like tabu search is often critically dependent on the quality of the solution used as a starting point for the algorithm. As the problem size increases, the solution space increases exponentially, and the choice of initial solutions becomes increasingly important. Tabu search is implemented for large sized problems in the multi-start mode. In this mode, tabu search is started from several initial tours which are widely separated in the solution space. The best tour obtained in all the runs is output by the algorithm. The preprocessing algorithm described in Section 2 can be easily tweaked to generate initial tours. To do this, the required number of initial tours are obtained from the tours present in the elite set $\mathscr{E}$ at the end of the preprocessing algorithm.

4 Computational Experience

In this section, we first describe the tabu search implementations that we create to test our preprocessing and initial tour generation schemes. We then report the experiment design and test beds of problems that we use for our experiments. Finally we present the results of our computational experiments.

Tabu search implementations: We combined our preprocessing method and initial tour generation method into three tabu search implementations, labeled A through C. Each implementation is defined as a combination of the preprocessing method used, method used to generate initial tours, and the

implementation of tabu search used. Two implementations of tabu search described in Basu et al. (2008) were considered; the TS-CI implementation which is the conventional implementation designed for tabu search on instances defined on complete graphs, and the TS-SAG implementation designed for tabu search on instances defined on sparse asymmetric graphs. In conventional implementations, non-existent arcs in a graph are represented as infinite cost arcs. Hence even though the neighborhood of a tour is much smaller for an ATSP instance defined on a sparse graph than one defined on a complete graph with the same number of nodes, conventional implementations of tabu search actually search a complete graph in the both cases. TS-SAG uses special data structures to eliminate the need for infinite cost arcs and so tabu search actually searches a much smaller neighborhood. This speeds up the TS-SAG algorithm significantly compared to TS-CI on ATSPs defined on sparse graphs. Table 1 describes the implementations that we use for our computational experiments. All the implementations

Table 1: Details of implementations

Implementation Scheme	**Preprocessing Scheme**	**Initial Solution**	**Tabu Search Implementation**
A	None	Generated randomly	TS-CI
B	None	From Section 3	TS-CI
C	From Section 2	From Section 3	TS-SAG

were coded in C, were run on a computer with an Intel Quad Core 2.4GHz processor and 3 GB of RAM. The length of the tabu list in all tabu search implementations was fixed at 8.

Comparisons between different implementations allow us to comment on the usefulness of the preprocessing method and initial tour generation method. A comparison of qualities of the tours obtained by implementations A and B shows us the effectiveness of the use of special methods to generate initial tours for multi-start tabu search. A comparison between implementations B and C shows us the usefulness of the preprocessing scheme combined with the use of special tabu search implementation designed for ATSPs defined on sparse graphs.

Test beds and computational experiments: We performed our experiments on randomly generated ATSP instances as well as on benchmark ATSP instances. Each of the instances was taken as described on a complete digraph. The randomly generated instances consisted of ten problem instances each of size 200, 300, 400, 500, and 600. The arc costs were chosen as integers randomly in the interval $[1, 1000]$. The benchmark instances consisted of 25 ATSP instances from Johnson et al. (2002) with 100 nodes

or more. Note that these instances include all similar instances in TSPLIB (Reinelt, 1991).

Our computational experiments were divided into two parts. In the first part we examined the performance of the three implementations when tabu search was allowed to run for 1000 iterations in each implementation. The performance parameters used to compare the implementations in this part for the randomly generated instances were (a) the average of the costs of the tours output by the implementation on all ten instances of a given size, and (b) the average of the execution times required by the implementation over all ten instances of a given size. For benchmark problem instances, the performance parameters were the cost of the tour output by the implementation for the instance, and the execution time taken by the implementation.

The second part deals with the performance of the three tabu search implementations when the execution time was fixed. We fixed the execution time for different problem sizes ensuring that a sufficient number of tabu search iterations are possible within the specified time limit. We defined t_k as the execution time that implementation A required to first encounter the tour it output after 1000 iterations on a ATSP instance of size k. Then for ATSPs of size s, we allot an execution time limit of 1000 t_s/t_{500} seconds. The performance measure used in this part is simply the average of the costs of the tours output by a given implementation on all ten instances of a particular size for randomly generated problems, and the cost of the tour output by a given implementation on a benchmark problem instance.

Computational results (first part): We first present the average of the costs of tours output by the three implementations on randomly generated ATSP instances in Table 2. In Table 3 we present the time required on average by the three implementations to run 1000 tabu search iterations on problems of a given size. The execution time is broken up into two parts, the time for preprocessing and the time for executing tabu search on the preprocessed instance. The time required to generate initial tours is included in the time for preprocessing, since the initial tours are generated as a by-product of the preprocessing operation itself.

Table 2: Average of the cost of tours output by the three implementations at the end of 1000 tabu search iterations

	200	**300**	**400**	**500**	**600**
A	40520.70	61666.70	85110.10	107101.60	131068.90
B	19909.20	37726.10	58923.10	80903.00	103893.20
C	20017.00	38628.30	59630.70	82305.40	106046.30

From Table 2 we see that the tours output by implementations B and C

Table 3: Execution time in seconds required by the three implementations to complete 1000 tabu search iterations

		200	**300**	**400**	**500**	**600**
A	Avg. preproc. time	0.00	0.00	0.00	0.00	0.00
	Avg. of time for TS	43.16	127.94	277.31	497.39	821.05
	Avg. of total time	43.16	127.94	277.31	497.39	821.05
B	Avg. preproc. time	117.40	264.10	469.20	732.70	1052.50
	Avg. of time for TS	42.32	131.77	287.32	533.45	873.23
	Avg. of total time	159.72	395.87	756.52	1266.15	1925.73
C	Avg. preproc. time	117.40	264.10	469.20	732.70	1052.50
	Avg. of time for TS	3.14	7.44	12.83	19.69	29.18
	Avg. of total time	120.54	271.54	482.03	752.39	1081.68

were better than those output by implementation A. Implementation B produced slightly better tours than implementation C although the tour costs were not significantly different when tested at a significance level of 0.01 (in a paired-t test). From Table 3 we see that as expected, the time required by tabu search in implementation C is much less than that required by tabu search in implementations A and B. Overall, implementation C required less time than implementation B, with the difference in the total time required increases with increasing problem size. Based on these observations, implementation C seems to dominate other implementations for randomly generated ATSP instances.

Table 4 summarizes the quality of tours output by the three implementations on the 25 benchmark ATSP instances. Since the costs of optimal tours for these problems are very different, we expressed the quality of tours output by the three implementations as a multiple of the Held-Karp lower bound (see Held and Karp, 1970, 1971) for that ATSP instance. Observe that implementations B and/or C produced the least cost tours in 19 out of the 25 instances. The four problems in the rgb class form a notable exception, implementation A generated the best tours to these problem instances. Table 5 presents the execution times required by the implementations on the benchmark problem instances. Between implementations B and C which provide the best quality tours in most cases, implementation C requires much less time, and is hence the preferred implementation. Notice that here too, the difference between the execution times of implementations B and C increases with increasing problem size.

Computational results (second part): Recall that in the second part of our experiments, we allowed each implementation to run for a pre-specified duration, and compared the quality of tours output by the implementation at the end of that duration. Table 6 presents the average of the costs of

Table 4: Costs of tours output by the implementations as a multiple of Held-Karp bound at the end of 1000 tabu search iterations

		Implementation		
Instance	**Size**	**A**	**B**	**C**
atex8	600	5.39	5.18	5.10
big702	702	5.22	5.30	5.36
dc112	112	1.01	1.01	1.01
dc126	126	1.00	1.00	1.03
dc134	134	1.01	1.01	1.02
dc176	176	1.02	1.01	1.03
dc188	188	1.01	1.01	1.02
dc563	563	1.05	1.05	1.08
dc849	849	1.05	1.05	1.05
dc895	895	1.02	1.02	1.13
dc932	932	1.01	1.01	1.11
ftv100	100	2.73	2.02	2.02
ftv110	110	2.78	2.16	2.29
ftv120	120	2.67	2.20	2.20
ftv130	130	3.11	2.25	2.25
ftv140	140	3.20	2.70	2.71
ftv150	150	3.34	2.27	2.45
ftv160	160	3.45	2.64	2.71
ftv170	170	3.64	2.89	2.85
kro124p	124	1.34	1.25	1.25
rbg323	323	2.86	3.16	3.16
rbg358	358	3.55	4.21	4.21
rbg403	403	2.11	2.48	2.48
rbg443	443	2.05	2.43	2.43
td100_1	100	1.32	1.12	1.12

the tours output by the different implementations over the ten randomly generated ATSP instances of a given size. The trends in the results from these experiments closely follow their counterparts in the first set. Here too, implementations B and C produced the least cost tours output and the difference in the quality of tours output by implementations B and C is not statistically significant.

For benchmark problem instances, we increased the time limit to ensure that tabu search could run from at least three initial tours for each of the instances. These allowable time limits were made proportional to the problem size. The execution times for the 25 instances are given in Table 7.

Table 8 reports the costs of tours output by the implementations as multiples of the Held-Karp bound for the corresponding problem. We see that implementations B and/or C produced the best tours in 21 of the 25 benchmark instances. Implementation A was seen to outperform implementations

Table 5: Execution time in seconds required by the three implementations to complete 1000 tabu search iterations on benchmark instances

		Implementation		
Instance	**Size**	**A**	**B**	**C**
atex8	600	848.9	1911.9	1176.4
big702	702	1188.6	2715.9	1538.5
dc112	112	8.1	44.7	38.6
dc126	126	10.4	56.8	49.0
dc134	134	12.9	66.8	55.7
dc176	176	25.9	114.3	96.3
dc188	188	31.6	135.3	110.7
dc563	563	675.0	1482.0	957.3
dc849	849	2139.1	4104.3	2213.2
dc895	895	2541.1	5098.3	2420.5
dc932	932	2978.9	5482.3	2625.1
ftv100	100	6.8	37.1	32.2
ftv110	110	9.3	44.6	38.2
ftv120	120	14.7	53.7	47.2
ftv130	130	12.9	64.4	57.2
ftv140	140	15.6	73.6	63.7
ftv150	150	20.1	88.1	75.6
ftv160	160	24.0	99.9	83.0
ftv170	170	29.1	111.5	95.0
kro124p	124	6.4	36.1	34.4
rbg323	323	138.9	450.8	323.8
rbg358	358	319.0	618.5	466.4
rbg403	403	208.7	693.1	503.0
rbg443	443	351.8	880.1	607.9
td100_1	100	6.6	38.2	35.4

Table 6: Average of the cost of tours output by the three implementations at the end of a pre-specified execution time

	200	**300**	**400**	**500**	**600**
A	39977.60	61282.20	85110.10	107101.60	131616.50
B	19819.50	37726.10	59032.90	81428.80	104865.30
C	19915.00	37993.60	58871.10	81610.10	105054.20

B and C for some of the dc instances. However, the difference in tour quality obtained from implementation A for these instances is only marginally better than those produced by implementations B and C.

In summary therefore, we observe that the preprocessing scheme and the initial tour generation scheme described in this paper combined with

Table 7: Execution times for benchmark problems

Problem	Time Limit	Problem	Time Limit
atex8	2000	ftv120	70
big702	3000	ftv130	80
dc112	50	ftv140	90
dc126	75	ftv150	100
dc134	80	ftv160	110
dc176	120	ftv170	120
dc188	140	kro124p	50
dc563	1500	rbg323	1000
dc849	4500	rbg358	1100
dc895	5500	rbg403	1200
dc932	7000	rbg443	1300
ftv100	50	td100_1	50
ftv110	60		

the TS-SAG implementation of tabu search produces results for moderate sized ATSP instances which are superior to conventional tabu search implementations for this problem.

5 Summary

In this paper, we suggest a tabu search implementation to solve asymmetric traveling salesman problems (ATSPs). In our implementation, we first use a cross entropy method based preprocessing algorithm to reduce the density of the graph describing the problem instance, and to generate good starting tours for tabu search to operate in a multi-start mode. We then use a tabu search implementation specially designed to solve ATSPs defined on sparse graphs.

We created three implementations, one without any of the enhancements suggested in the paper, a second in which only initial tours were generated with our method and a third in which the graph was preprocessed and initial tours were generated, both by the method we suggested in this paper. The special tabu search implementation designed for sparse graphs from Basu et al. (2008) could only be used in the third implementation. We compared the implementations based on extensive computational experiments using randomly generated ATSP instances and benchmark problem instances. We performed two types of experiments, one in which we fixed the number of tabu search iterations, and another in which we fixed the total execution time. From our experiments we concluded that the best tabu search implementation for medium sized ATSPs is the one in which both the preprocessing and the initial tour generation are done based on the cross

Table 8: Costs of tours output by the implementations as a multiple of Held-Karp bound at the end of the pre-specified execution time

		Implementation		
Instance	**Size**	**A**	**B**	**C**
atex8	600	5.51	5.25	5.05
big702	702	12.65	4.91	5.19
dc112	112	1.01	1.01	1.01
dc126	126	1.00	1.00	1.02
dc134	134	1.01	1.01	1.02
dc176	176	1.02	1.01	1.03
dc188	188	1.00	1.01	1.02
dc563	563	1.05	1.05	1.08
dc849	849	1.05	1.05	1.05
dc895	895	1.03	1.03	1.13
dc932	932	1.00	1.01	1.11
ftv100	100	2.65	1.90	1.90
ftv110	110	2.78	1.93	1.93
ftv120	120	2.67	2.16	2.16
ftv130	130	3.04	2.30	2.30
ftv140	140	3.20	2.42	2.52
ftv150	150	3.34	2.59	2.59
ftv160	160	3.45	2.78	2.68
ftv170	170	3.64	2.85	2.84
kro124p	124	1.34	1.25	1.25
rbg323	323	3.61	2.19	2.20
rbg358	358	859.84	2.72	2.78
rbg403	403	405.68	1.69	1.74
rbg443	443	367.65	1.74	1.79
td100_1	100	1.32	1.13	1.12

entropy method, followed by the TS-SAG tabu search implementation (see Basu et al., 2008) specially designed for performing tabu search on ATSPs defined on sparse graphs.

References

Applegate D, Bixby R, Chvatal V and Cook W (2006). *The Traveling Salesman Problem: A Computational Study.* Princeton University Press.

Basu S, Gajulapalli R and Ghosh D (2008). *Implementing tabu serach to exploit sparsity in atsp instances.* Working Paper Series, Indian Institute of Management Ahmedabad, 2008-10-02.

Basu S and Ghosh D (2008). *A review of the tabu search literature on*

traveling salesman problems. Working Paper Series, Indian Institute of Management Ahmedabad, W.P. No. 2008-10-01.

Boer P, Kroese D, Mannor S and Rubinstein R (2005). A tutorial on the cross-entropy method. *Annals of Operations Research* **134**: 19–67.

Chepuri K and Homem-de Mello T (2005). Solving the vehicle routing problem with stochastic demands using the cross entropy method. *Annals of Operations Research* **134**: 193–181.

Cirassela J, Johnson D, McGeoch L and Zhang W (2001). The asymmetric traveling salesman problem: algorithms, instance generators, and tests. In: Buchsbaum A and Snoeyink J (eds). *Algorithm Engineering and Experimentation*. Third International Workshop, ALENEX 2001, Lecture Notes in Computer Science **2153**, pp 32–59. Springer-Verlag.

Fischetti M, Lodi A and Toth P (2002). Exact methods for asymmetric traveling salesman problem. In: Gutin G and Punnen P (eds). *The Traveling Salesman Problem and Its Variations* **4**, pp 169–206. Kluwer Academic Publisher: London.

Glover F and Laguna M (1998). *Tabu Search*. Kluwer Academic Publisher: London.

Goossens J and Baruah S (2001). Multiprocessor preprocessing algorithms for uniprocessor on-line scheduling. In: *The 21th International Conference on Distributed Computing Systems*.

Held M and Karp R (1970). The traveling salesman problem and minimum spanning trees. *Operations Research* **18**: 1138–1162.

Held M and Karp R (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming* **1**: 6–25.

Johnson D, Gutin G, McGeoch L, Yeo A, Zhang W and Zverovich A (2002). Experimental analysis of heuristics for the ATSP. In: Gutin G and Punnen P (eds). *The Traveling Salesman Problem and Its Variations* **10**, pp 445–488. Kluwer Academic Publishers: London.

Karp R (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, pp 85–103. Plenum Press.

Khumawala B (1975). An efficient branch and bound algorithm for the warehouse location problem. *Management Science* **18**: B718–B731.

Reinelt G (1991). TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing* **3**: 376–384.

Rubinstein R (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research* **99**: 89–112.

Rubinstein R (1999). The simulated entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability* **2**: 127–190.

Rubinstein R (2001). Combinatorial Optimization, Cross-Entropy, Ants and Rare Events. In: Uryasev S and Pardalos P M (eds). *Stochastic optimization: algorithms and applications*, pp 445–488. Kluwer Academic Publishers: London.

Toth P and Vigo D (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing* **15**: 333–346.