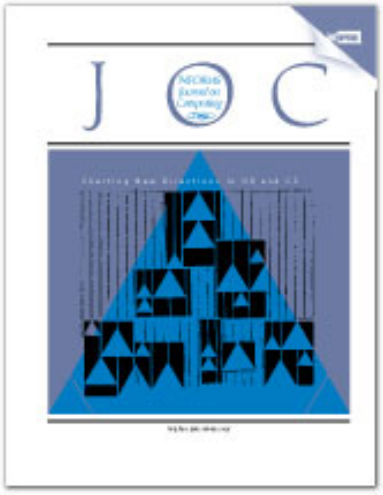

ORSA Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

TSPLIB—A Traveling Salesman Problem Library

Gerhard Reinelt,

To cite this article:
Gerhard Reinelt, (1991) TSPLIB—A Traveling Salesman Problem Library. ORSA Journal on Computing 3(4):376-384. http://dx.doi.org/10.1287/ijoc.3.4.376

ORSA Journal on Computing
Vol. 3, No. 4, Fall 1991

0899-1499/91/0304-0376 $01.25

TSPLIB—A Traveling Salesman Problem Library

GERHARD REINELT *Institut für Mathematik, Universität, Augsburg, Universitätsstrasse 8, D-8900 Federal Republic of Germany, CSNET: reinelt@augsopt.uni-augsburg.de*

(Received: February 1991; accepted: May 1991)

This paper contains the description of a traveling salesman problem library (TSPLIB) which is meant to provide researchers with a broad set of test problems from various sources and with various properties. For every problem a short description is given along with known lower and upper bounds. Several references to computational tests on some of the problems are given.

Maybe the by far most attacked combinatorial optimization problem is the Traveling Salesman Problem (TSP). There is a vast amount of literature on it and many computational experiments are reported. Unfortunately, there are only a few commonly distributed sample problems, as e.g. the 120-city and 666-city problems by M. Grötschel (cf. [10] and [12]) or the 2392-city problems by M. Padberg and G. Rinaldi (cf. [17]). This paper introduces a library of sample instances for the TSP (and related problems) from various sources and of various types. At present, instances of the following problem classes are available.

Symmetric traveling salesman problem (TSP). Given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimum total length visiting each node exactly once. The distance from node i to node j is the same as from node j to node i.

Asymmetric traveling salesman problem (ATSP). Given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimum total length visiting each node exactly once. In this case the distance from node i to node j and the distance from node j to node i may be different.

Capacitated vehicle routing problem (CVRP). We are given $n - 1$ nodes, one depot and distances from the nodes to the depot, as well as between nodes. All nodes have demands which can be satisfied by the depot. For delivery to the nodes, trucks with identical capacities are available. The problem is to find tours for the trucks of minimum total length that satisfy the node demands without violating truck capacity constraint. The number of trucks is not specified. Each tour visits a subset of the nodes and starts and terminates at the depot. (Remark: In some data files a collection of alternate depots is given. A CVRP is then given by selecting one of these depots.)

In the current version all problems are defined on a complete graph, but problems on sparse graphs can also be specified. Moreover, there is a possibility to require that certain edges appear in the solution of a problem.

1. The File Format

It is our intention not only to distribute problem data but also solutions, Lagrange multipliers or special subgraphs that might be of interest. The file format was designed with this goal in mind. In this section we describe the basic version. Future enhancements (especially for the definition of other problem classes) are possible.

Each file consists of basically two parts: a specification part and a data part. The specification part contains information on the file format and on its contents. The data part contains explicit data.

1.1. The Specification Part

All entries in this section are of the form ⟨*keyword*⟩:⟨*value*⟩, where ⟨*keyword*⟩ denotes an alphanumerical keyword and ⟨*value*⟩ denotes alphanumerical or numerical data. The ":" is optional and may be replaced by a blank character. The terms ⟨*string*⟩, ⟨*integer*⟩ and ⟨*real*⟩ denote character string, integer or real data, respectively. The order of specification of the keywords in the data file is arbitrary (in principle), but must be consistent, i.e., whenever a keyword is specified all necessary information for the correct interpretation of the keyword has to be known.

Below we give a list of all available keywords.

1.1.1. NAME:⟨*string*⟩

Identifies the data file.

1.1.2. TYPE:⟨*string*⟩

Specifies the nature of the data. This is sometimes necessary to allow for a correct interpretation of the data.

Subject classifications: Networks/Graphs: Traveling Salesman: A data base of test problems instances.
Other key words: Traveling salesman problem, a data base of test problem instances.

Possible types are

TSP	Data for a symmetric traveling salesman problem
ATSP	Data for an asymmetric traveling salesman problem
CVRP	Capacitated vehicle routing problem data
TOUR	A collection of tours
GRAPH	The file specifies a graph
MULT	The file contains Lagrange node multipliers.

1.1.3. COMMENT:⟨*string*⟩

Additional comment(s) on the data.

1.1.4. DIMENSION:⟨*integer*⟩

For a TSP or ATSP, the dimension is the number of its nodes. For a CVRP, it is the total number of nodes and depots.

1.1.5. CAPACITY:⟨*integer*⟩

Specifies the truck capacity in a CVRP.

1.1.6. GRAPH_TYPE:⟨*string*⟩

If the underlying graph is not complete, this entry must be specified. If left unspecified, a complete graph is assumed.

COMPLETE_GRAPH	The underlying graph is complete
SPARSE_GRAPH	The file contains a sparse subgraph.

If the graph is not complete then usually it has to be known implicitly to what master graph the graph data is related

1.1.7. EDGE_TYPE:⟨*string*⟩

Specifies whether the edges are directed or undirected. If left unspecified an undirected graph is assumed. The values are

DIRECTED	The edges (arcs) are directed
UNDIRECTED	The edges are not directed.

1.1.8. EDGE_WEIGHT_TYPE:⟨*string*⟩

Specifies how the edge "weights" (or "lengths") are given if they are given explicitly. The values are

EXPLICIT	Weights are listed explicitly in the corresponding section
EUC_2D	Weights are given by the 2-dimensional Euclidean distance
EUC_3D	Weights are given by the 3-dimensional Euclidean distance
MAX_2D	Weights are given by the 2-dimensional maximum distance
MAX_3D	Weights are given by the 3-dimensional maximum distance
MAN_2D	Weights are given by the 2-dimensional Manhattan distance
MAN_3D	Weights are given by the 3-dimensional Manhattan distance
GEO	Weights are given by the geographical distance
ATT	Special distance function for problems att48 and att532
XRAY1	Special distance function for crystallography problems (Version 1)
XRAY2	Special distance function for crystallography problems (Version 2)
SPECIAL	There is a special distance function documented elsewhere.

1.1.10. EDGE_WEIGHT_FORMAT:⟨*string*⟩

Describes the format of the edge weights if they are given explicitly. The values are

FUNCTION	Weights are given by a function (see above)
FULL_MATRIX	Weights are given by a full matrix
UPPER_ROW	Upper triangular matrix (row-wise without diagonal entries)
LOWER_ROW	Lower triangular matrix (row-wise without diagonal entries)
UPPER_DIAG_ROW	Upper triangular matrix (row-wise including diagonal entries)
LOWER_DIAG_ROW	Lower triangular matrix (row-wise including diagonal entries)
UPPER_COL	Upper triangular matrix (column-wise without diagonal entries)
LOWER_COL	Lower triangular matrix (column-wise without diagonal entries)
UPPER_DIAG_COL	Upper triangular matrix (column-wise including diagonal entries)
LOWER_DIAG_COL	Lower triangular matrix (column-wise including diagonal entries)
WEIGHT_LIST	Weights are specified in a list.

1.1.11. EDGE_DATA_FORMAT:⟨*string*⟩

Specifies how edges are given, if the file contains a graph that is not complete. The values are

ADJ_LIST	The graph is given as an adjacency list
EDGE_LIST	The graph is given by an edge list.

The default value is COMPLETE, i.e., no explicit edge data is given.

1.1.12. NODE_TYPE:⟨*string*⟩

Specifies whether the nodes have weights associated with them. If left unspecified, unweighted nodes are assumed. The values are

WEIGHTED_NODES	The nodes have associated weights
UNWEIGHTED_NODES	The nodes are unweighted

1.1.13. NODE_COORD_TYPE:⟨*string*⟩

Specifies whether coordinates are associated with each node (which, for example may be used for either graphical display or distance computations). The values are

TWOD_COORDS	The nodes have 2-dimensional coordinates
THREED_COORDS	The nodes have 3-dimensional coordinates
NO_COORDS	The nodes do not have associated coordinates

The default value is NO_COORDS.

Coordinates may also receive offsets and scalings. These data may be used for modeling the possibly different speeds of a machine in the x- and y-directions. The following six keywords specify offset and scaling for the coordinates.

1.1.14. COORD1_OFFSET:⟨*real*⟩

1.1.15. COORD1_SCALE:⟨*real*⟩

1.1.16. COORD2_OFFSET:⟨*real*⟩

1.1.17. COORD2_SCALE:⟨*real*⟩

1.1.18. COORD3_OFFSET:⟨*real*⟩

1.1.19. COORD3_SCALE:⟨*real*⟩

The offsets (default values 0.0) specify numbers that are added to the respective coordinates. After addition of the offsets the coordinates are multiplied by the respective scaling factors (default values 1.0).

1.1.20. DISPLAY_DATA_TYPE:⟨*string*⟩

Specifies how a graphical display of the nodes can be obtained. The values are

COORD_DISPLAY	Display is generated from the node coordinates
TWOD_DISPLAY	Explicit 2-dimensional coordinates are given
NO_DISPLAY	No graphical display is possible

The default value is NO_DISPLAY.

1.1.21. EOF:

Terminates the input data. This entry is optional.

1.2 The Data Part

Depending on the choice of specifications some additional data (in specified format) may be required. These data are given in corresponding data sections following the specification part. Each data section is begun by the corresponding keyword. The length of the section is either implicitly known from the format specification, or the section is terminated by an appropriate end-of-section identifier.

1.2.1. NODE_COORD_SECTION:

Node coordinates are given in this section. Each line is of the form

⟨*integer*⟩⟨*real*⟩⟨*real*⟩

if NODE_COORD_TYPE is TWOD_COORDS, or

⟨*integer*⟩⟨*real*⟩⟨*real*⟩⟨*real*⟩

if NODE_COORD_TYPE is THREED_COORDS. The integers give the number of the respective nodes. The real numbers give the associated coordinates.

1.2.2.. DEPOT_SECTION:

Contains a list of possible alternate depot nodes. This list is terminated by a -1.

1.2.3. DEMAND_SECTION:

The demands of all nodes of a CVRP are given in the form (per line)

⟨*integer*⟩⟨*integer*⟩

The first integer specifies a node number, the second its demand. The depot nodes must also occur in this section. Their demands are 0.

1.2.4. FIXED_EDGES_SECTION:

In this section edges are listed that are required to appear in each solution to the problem. The edges to be fixed are given in the form (per line)

⟨*integer*⟩⟨*integer*⟩

meaning that the edge (arc) from the first node to the second node has to be contained in a solution. This section is terminated by a -1.

1.2.5. DISPLAY_DATA_SECTION:

If DISPLAY_DATA_TYPE is TWOD_DISPLAY, the 2-dimensional coordinates from which a display can be generated are given in the form (per line)

⟨*integer*⟩⟨*real*⟩⟨*real*⟩

The integers specify the respective nodes and the real numbers give the associated coordinates.

1.2.6. NODE_WEIGHT_SECTION:

The nonzero node weights are given in the form (per line)

⟨*integer*⟩⟨*real*⟩

The integers specify the respective nodes and the real numbers give the associated node weight. This section is terminated by a -1.

1.2.7. TOUR_SECTION:

A collection of tours is specified in this section. Each tour is given by a list of integers giving the sequence in which the nodes are visited in this tour. Every such tour is

terminated by a -1. An additional -1 terminates this section.

1.2.8. EDGE_DATA_SECTION:

Edges of a graph are specified in either of the two formats allowed in the EDGE_DATA_TYPE entry. If the type is EDGE_LIST, then the edges are given as a sequence of lines of the form

$$\langle integer\rangle\langle integer\rangle$$

each entry giving the terminal nodes of some edge. The list is terminated by a -1. If the type is ADJ_LIST, the section consists of a list of adjacency lists for nodes. The adjacency list of a node x is specified as

$$\langle integer\rangle\langle integer\rangle \ldots \langle integer\rangle - 1$$

where the first integer gives the number of node x and the following integers (terminated by -1) the numbers of nodes adjacent to x. The list of adjacency lists is terminated by an additional -1.

1.2.9. EDGE_WEIGHT_SECTION:

The edge weights are given in the format specified by the EDGE_WEIGHT_FORMAT entry. The matrix formats are self-explanatory, with implicitly known lengths. If the format WEIGHT_LIST is specified then a sequence of lines of the form

$$\langle integer\rangle\langle integer\rangle\langle integer\rangle$$

must be given where the first two integers give the terminal nodes of the edge and the third integer its edge weight. The list is terminated by a -1.

2. The Distance Functions

In this section we describe the specific edge weight computations listed under the keyword EDGE_WEIGHT_TYPE. In each case we give a (simplified) C-implementation for computing the distances from the input coordinates. All computations involving floating-point numbers are carried out in double precision arithmetic. The integers are assumed to be represented in 32-bit words. Since distances are required to be integral, some rounding may be necessary: we have used the C truncation function "nint." Explicitly specified distances must be integral.

2.1. Euclidean Distance (L_2-Metric)

For edge weight type EUC_2D and EUC_3D, floating point coordinates must be specified for each node. Let x[i], y[i], and z[i] be the coordinates of node i.

In the 2-dimensional case the distance between two points i and j is computed as follows:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
dij = nint(sqrt(xd*xd + yd*yd) + 0.5);
```

In the 3-dimensional case we have:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
zd = z[i] - z[j];
dij = nint(sqrt(xd*xd + yd*yd + zd*zd) + 0.5);
```

"Sqrt" is the C square root function.

2.2 Manhattan Distance (L_1-Metric)

Distances are given as Manhattan distances if the edge weight type is MAN_2D or MAN_3D. They are computed as follows.

Two-dimensional case:

```
xd = abs(x[i] - x[j]);
yd = abs(j[i] - y[j]);
dij = nint(xd + yd + 0.5).
```

3-dimensional case:

```
xd = abs(x[i] - x[j]);
yd = abs(y[i] - y[j]);
zd = abs(z[i] - z[j]);
dij = nint(xd + yd + zd + 0.5).
```

Note that a scaling factor for the coordinates may be given in the input data file. This factor must be applied to the coordinates before doing the computations.

2.3. Maximum Distance (L_∞-Metric)

Maximum distances are computed if the edge weight type is MAX_2D or MAX_3D.

Two-dimensional case:

```
xd = abs(x[i] - x[j]);
yd = abs(y[i] - y[j]).
dij = max(nint(xd + 0.5), nint(yd + 0.5));
```

Three-dimensional case:

```
xd = abs(x[i] - x[j]);
yd = abs(y[i] - y[j]);
zd = abs(z[i] - z[j]);
dij = max(nint(xd + 0.5), nint(yd + 0.5),
    nint(zd + 0.5)).
```

If scaling factors are present, they must be applied before the distance computations.

2.4. Geographical Distance

If the traveling salesman problem is a geographical problem, then the nodes correspond to points on the earth and the distance between two points is their distance on the idealized sphere with radius 6378.388 kilometers. The node coordinates give the geographical latitude and longitude of the corresponding point on the earth. Latitude and longitude are given in the form DDD.MM where DDD are the degrees and MM the minutes. A positive latitude is assumed to be "North," negative latitude means "South." Positive

Table I
List of Problems

Name	No. of Cities	Type	Bounds	References
ali535	535	GEO	202310	[17]
att48	48	ATT	10628	[17]
att532	532	ATT	27686	[5, 6, 9, 12, 15, 17, 19, 20]
bayg29	29	GEO	1610	[11]
bays29	29	GEO	2020	[11]
bier127	127	EUC_2D	118282	
d198	198	EUC_2D	15780	[18]
d493	493	EUC_2D	35002	
d657	657	EUC_2D	48912	
d1291	1291	EUC_2D	[50209,51677]	[18]
d1655	1655	EUC_2D	[61544,63099]	[18]
d2103	2103	EUC_2D	[78276,81001]	[18]
dantzig42	42	MATRIX	699	[4, 12]
dsj1000	1000	EUC_2D	[1861690,1880480]	
eil51	51	EUC_2D	426	[17]
eil76	76	EUC_2D	538	[17]
eil101	101	EUC_2D	629	[17]
fl417	417	EUC_2D	11861	
fl1400	1400	EUC_2D	[19783,20280]	
fl1577	1577	EUC_2D	[22134,22371]	[18]
fl3795	3795	EUC_2D	[26657,29192]	
gil262	262	EUC_2D	2378	[17]
gr17	17	MATRIX	2085	[12]
gr21	21	MATRIX	2707	[12]
gr24	24	MATRIX	1272	[12]
gr48	48	MATRIX	5046	[12, 21, 22]
gr96	96	GEO	55209	[12]
gr120	120	MATRIX	6942	[1, 8, 10, 20, 22]
gr137	137	GEO	69853	[12]
gr202	202	GEO	40160	[12]
gr229	229	GEO	134602	[12]
gr431	431	GEO	171414	[12]
gr666	666	GEO	294358	[8, 12, 20]
hk48	48	MATRIX	11461	[12, 13, 17]
kroA100	100	EUC_2D	21282	[7, 8, 12, 17, 22]
kroB100	100	EUC_2D	22141	[7, 8, 12, 17, 22]
kroC100	100	EUC_2D	20749	[7, 8, 12, 17, 22]
kroD100	100	EUC_2D	21294	[7, 8, 12, 17, 22]
kroE100	100	EUC_2D	22068	[7, 8, 12, 17, 22]
kroA150	150	EUC_2D	26524	[7, 17]
kroB150	150	EUC_2D	26130	[7, 17]
kroA200	200	EUC_2D	29368	[7, 17]
kroB200	200	EUC_2D	29437	[7, 17]
lin105	105	EUC_2D	14379	[17]
lin318	318	EUC_2D	42090	[3, 14, 17, 20, 22]
linph318	318	EUC_2D	41345	[17]
p654	654	EUC_2D	34643	[18]
pcb442	442	EUC_2D	50778	[5, 6, 8, 9, 12, 18, 20–22]
pcb1173	1173	EUC_2D	56892	[18]

Table I—*Continued*

Name	No. of Cities	Type	Bounds	References
pcb3038	3038	EUC_2D	[136588,139667]	[18]
pr76	76	EUC_2D	108159	[17]
pr107	107	EUC_2D	44303	[17]
pr124	124	EUC_2D	59030	[17]
pr136	136	EUC_2D	96772	[17]
pr144	144	EUC_2D	58537	[17]
pr152	152	EUC_2D	73682	[17]
pr226	226	EUC_2D	80369	[17]
pr264	264	EUC_2D	49135	[17]
pr299	299	EUC_2D	48191	[17]
pr439	439	EUC_2D	107217	[17]
pr1002	1002	EUC_2D	259045	[17]
pr2392	2392	EUC_2D	378032	[17, 18]
rat99	99	EUC_2D	1211	
rat195	195	EUC_2D	2323	
rat575	575	EUC_2D	6773	
rat783	783	EUC_2D	8806	
rd100	100	EUC_2D	7910	[18]
rd400	400	EUC_2D	[15245,15311]	[18]
rl1304	1304	EUC_2D	[224325,259940]	
rl1323	1323	EUC_2D	[265815,274119]	[18]
rl1889	1889	EUC_2D	[311705,320340]	[18]
rl5915	5915	EUC_2D	[522541,574596]	[18]
rl5934	5934	EUC_2D	[542573,565744]	[18]
rl11849	11849	EUC_2D	[856950,940843]	
st70	70	EUC_2D	675	[12, 17]
u159	159	EUC_2D	42080	
u574	574	EUC_2D	36905	
u724	724	EUC_2D	41910	
u1060	1060	EUC_2D	[222651,226665]	[18]
u1432	1432	EUC_2D	[152535,154698]	[18]
u1817	1817	EUC_2D	[51332,58602]	
u2152	2152	EUC_2D	[63659,65604]	
u2319	2319	EUC_2D	[232424,235503]	
vm1084	1084	EUC_2D	[210245,243693]	[18]
vm1748	1748	EUC_2D	[332061,340926]	

longitude means "East," negative latitude is assumed to be "West." For example, the input coordinates for Augsburg are 48.23 and 10.53, meaning 48.23 North and 10.53 East.

Let `x[i]` and `y[i]` be coordinates for city i in the above format. First the input is converted to geographical latitude and longitude given in radians.

```
PI = 3.141592;
deg = nint(x[i]);
min = x[i] - deg;
latitude[i] = PI*(deg + 5.0 * min) / 3.0) / 180.0;
deg = nint(y[i]);
min = y[i] - deg;
longitude[i] = PI*(deg + 5.0 * min) / 3.0) /
               180.0;
```

The distance between two nodes i and j in kilometers is then computed as follows:

```
RRR = 6378.388;
q1 = cos(longitude[i] - longitude[j]);
q2 = cos(latitude[i] - latitude[j]);
q3 = cos(latitude[i] + latitude[j]);
dij = nint((RRR*acos(0.5*((1.0 + q1)*q2 - (1.0
      - q1)*q3)) + 1.0).
```

"`Acos`" is the C inverse-cosine function.

2.5. Pseudo-Euclidean Distance

The edge weight type `ATT` corresponds to a special "pseudo-Euclidean" distance function. Let `x[i]` and `y[i]` be the coordinates of node i. The distance between two

points i and j is computed as follows:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
rij = sqrt((xd*xd + yd*yd) / 10.0);
tij = nint(rij);
if (tij<rij) dij = tij + 1;
else dij = tij;
```

2.6. Distance for Crystallography Problems

We have included into TSPLIB the crystallography problems as described in [2]. These problems are not explicitly given but subroutines are provided to generate the 12 problems mentioned in this reference and subproblems thereof. More details can be found in section 3.2 of this paper.

To compute distances for these problems the movement of three motors has to be taken into consideration. There are two types of distance functions: one that assumes equal speed of the motors (XRAY1) and one that uses different speeds (XRAY2). The corresponding distance functions are given as FORTRAN implementations (files deq.f, resp. duneq.f) in the distribution file.

Since we would like all distances to be integral we propose to multiply the distances computed by the original subroutines by 100.0 and round to the nearest integer.

We list our modified distance function for the case of equal motor speeds in the FORTRAN version below.

```
INTEGER FUNCTION ICOST(V,W)
INTEGER V,W
DOUBLE PRECISION DMIN1,DMAX1,DABS
DOUBLE PRECISION DISTP,DISTC,DISTT,COST
DISTP = DMIN1(DABS(PHI(V) - PHI(W)),DABS(DABS
       (PHI(V) - PHI(W)) - 360.0E + 0))
DISTC = DABS(CHI(V) - CHI(W))
DISTT = DABS(TWOTH(V) - TWOTH(W))
COST = DMAX1(DISTP / 1.00,
      DISTC / 1.0,DISTT / 1.00)
C *** Make integral distances ***
ICOST = AINT(100.0*COST + 0.5)
RETURN
END
```

The numbers PHI(), CHI(), and TWOTH() are the respective x-, y-, and z-coordinates of the points in the generated traveling salesman problems.

Note that on the distribution tape you will find only the original distance computation without the above modification.

2.7. Verification

To verify correctness of the distance function implementations (which is not critical for the Manhattan or maximum distance) we give the length of some "canonical" tours $1, 2, 3, \ldots, n$.

The canonical tours for pcb442, gr666, and att532 have lengths 221 440, 423 710, and 309 636, respectively.

The canonical tour for the problem xray14012 (the 8th problem considered in [2]) with distance XRAY1 has length 15 429 219. With distance XRAY2 it has the length 12 943 294.

3. Description of the Library Files

In this section we give a list of all problems that are currently available and a short information on them. The bounds will be updated in regular intervals and are included as a data file of TSPLIB (see 3.3).

3.1. Symmetric Traveling Salesman Problems

Table I gives the problem names along with number of cities, problem type, known lower and upper bounds for the optimum tour length, and references to publications where the respective problem has been considered. The names of the corresponding data files are obtained by appending the suffix ".tsp" to the problem name.

Some optimum tours are also provided. The corresponding files have names with suffix ".opt.tour".

3.2. Crystallography Problems

In the file xray.problems we distribute the routines written by Bland and Shallcross and the necessary data to generate the crystallography problems discussed in [2]. More details can also be found there.

The file xray.problems is one file into which the single files mentioned in the sequel have been merged. These single files have to be extracted from xray.problems using an editor. The following original files are provided

```
read.me   deq.f  duneq.f  daux.f  gentsp.f
 a.data  b.data   d.data  e.data    f.data
```

In addition we have included specially prepared data files to generate the 12 problems mentioned in [2]. The files have the names xray1.data through xray12.data. Using these data files 12 symmetric TSPs can be generated using the program gentsp.f. We propose to name the respective problems instances xray4472, xray2950, xray7008, xray2762, xray6922, xray9070, xray5888, xray14012, xray5520, xray13804, xray14464, and xray13590.

To verify the correct use of the generating routines we list part of the file xray14012.tsp.

```
NAME:xray14012
COMMENT:Crystallography problem 8 (Bland /
          Shallcross)
```

```
TYPE:TSP
DIMENSION:14012
EDGE_WEIGHT_TYPE:XRAY2
NODE_COORD_SECTION
    1   91.802854544029  6.4097888697337  176.39830490027
    2   87.715643397938  6.4659384343446  165.56800324542
    3   83.587211962870  6.4895404648110  163.53828545043
    4   79.460007412434  6.4797580053949  165.86438271158
    :
    :
14009  100.539992581837  6.4797580053959  165.86438271158
14010   96.412788031401  6.4895404648110  163.53828545043
14011   92.284356596333  6.4659384343446  165.56800324542
14012   88.197145450242  6.4097888697337  176.39830490027
```

3.3. Capacitated Vehicle Routing Problems

Data for capacitated vehicle routing problems is given in data files whose names have the suffix ".vrp." At present, we have the data files

```
att48.vrp    eil30.vrp    eil7.vrp  eilB76.vrp    eil13.vrp
eil31.vrp  eilA101.vrp  eilC76.vrp   eil22.vrp    eil33.vrp
eilA76.vrp   eilD76.vrp   eil23.vrp   eil51.vrp  eilB101.vrp
gil262.vrp
```

3.4. Further Special Files

In addition to the data files the following special files are contained in the library.

TSPLIB_VERSION:	Gives the current version of the library
TSPLIB_VALUES:	Provides information on optimum solution values
TSPLIB_CHANGES:	Documents changes from one version to the next one
TSPLIB_CHANGES:	Documents changes from one version to the next one
READ_ME:	Description of the library
NOTICE:	A note on the distribution policy

4. Remarks

1. The problem lin318 is originally a Hamiltonian path problem. One obtains this problem by adding the additional requirement that the edge from 1 to 214 is contained in the tour. The data is given in linhp318.tsp.
2. Some data sets are referred to by different names in the literature. Below we give the corresponding names used in [17] and [12].

TSPLIB	[17]	[12]	TSPLIB	[17]	[12]
att48	ATT048	—	att532	ATT532	—
dantzig42	—	42	eil101	EIL10	—
eil51	EIL08	—	eil76	EIL09	—
gil262	GIL249	—	gr120	—	120
gr137	GH137	137	gr202	GH202	202
gr229	GH229	229	gr431	GH431	431
gr666	GH666	666	gr96	GH096	96

hk48	—	48H	kroA100	KRO124	100A
kroB100	KRO125	100B	kroC100	KRO126	100C
kroD100	KRO127	100D	kroE100	KRO128	100E
kroA150	KRO30	—	kroB150	KRO31	—
kroA200	KRO32	—	kroB200	KRO33	—
lin105	LK105	—	lin318	LK318	—
linhp318	LK318P	—	pr1002	TK1002	—
pr107	TK107	—	pr124	TK124	—
pr136	TK136	—	pr144	TK144	—
pr152	TK152	—	pr226	TK226	—
pr2392	TK2392	—	pr264	TK264	—
pr299	TK299	—	pr439	TK439	—
pr76	TK076	—	st70	KRO070	70

3. The vehicle routing problems are also available in a TSP version. Here the depots are just treated as normal nodes. The problem gil262 originally contained two identical nodes, of which one was eliminated.
4. Potential contributors to this library should provide their data files in the above format and contact: Gerhard Reinelt, Institut für Mathematik, Universität Augsburg, Universitätsstrasse 8, D-8900 Augsburg, Federal Republic of Germany, Tel: (821) 598 2212, Fax: (821) 598 2200, E-Mail: reinelt@augsopt.uni-augsburg.de.
5. Informations on new bounds or optimum solutions for library problems are also appreciated (to be included into the file TSPLIB_VALUES).

5. Access to the Library

TSPLIB is available via E-Mail either from NETLIB or from the Computer and Information Technology Institute, Rice University.

a) NETLIB

To get a description of the general NETLIB index use the following.

```
mail netlib@ornl.gov
send index
```

b) Computer and Information Technology Institute

To get a description of available data use

```
mail softlib@rice.edu
send README
send INDEX
send CATALOGUE
```

It is also possible to use anonymous `ftp` (`ftp softlib.rice.edu`) to get the library.

In case you have no access to E-Mail or you cannot obtain the library by any other means you can receive it from the author of this paper. In that case however, distribution is only possible on a streamer tape (DC6150 or equivalent) which will be written on a SUN SPARCstation. In addition, a shipping and handling fee (DM 120,– or US-$100,–) will be necessary (this will include the streamer tape). The current version of the library is TSPLIB 1.2.

ACKNOWLEDGMENT

Thanks are due to Giovanni Rinaldi for helpful discussions and for providing a substantial part of the problems and to Robert E. Bixby and the Center for Research on Parallel Computation for installing E-Mail access to the library. Further optimum solutions are known (personal communication with D. Applegate, R.E. Bixby, V. Chvátal and W. Cook). The values are contained in the file TSPLIB-Values of TSPLIB 1.2.

REFERENCES

1. P. Ablay, 1987. Optimieren mit Evolutionsstrategien, *Spektrum der Wissenschaft 7:* 104–115.
2. R.E. Bland and D.F. Shallcross, 1987. Large Travelling Salesman Problems Arising from Experiments in X-ray Crystallography: A Preliminary Report on Computation, Technical Report No. 730, School of OR/IE, Cornell University, Ithaca, NY.
3. H. Crowder and M.W. Padberg, 1980. Solution of Large-Scale Symmetric Travelling Salesman Problems to Optimality, *Management Science 26:* 495–509.
4. G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, 1954. Solution of a Large Scale Traveling-Salesman Problem, *Operations Research 2:* 393–410.
5. G. Dueck, 1990. New Simple Heuristics for Optimization, presented at the TSP-Workshop 1990, CRPC, Rice University.
6. G. Dueck and T. Scheuer, 1988. Threshold Accepting: A General Purpose Algorithm Appearing Superior to Simulated Annealing, IBM Scientific Center Heidelberg, TR 88.10.011.
7. W. Felts, P. Krolak and G. Marble, 1971. A Man-Machine Approach towards Solving the Travelling-Salesman-Problem, *Communications ACM 14:* 327–334.
8. B. Fruhwirth, 1987. Untersuchung über genetisch motivierte Heuristiken für das Euklidische Rundreiseproblem, Technischer Bericht 87-103, TU Graz.
9. M. Gorges-Scheuter, 1990. A Massive Parallel Genetic Algorithm for the TSP, presented at the TSP-Workshop 1990, CRPC, Rice University.
10. M. Grötschel, 1977. *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme*, Verlag Anton Hain, Meisenheim am Glan.
11. M. Grötschel, 1985 Optimierungsmethoden I, Lecture Notes, Universität Ausgburg.
12. M. Grötschel and O. Holland, 1989. Solution of Large-Scale Symmetric Travelling Salesman Problems, Report No. 73 Schwerpunktprogramm der Deutschen Forschungsgemeinschaft, Universität Augsburg, Augsburg (to appear in *Mathematical Programming*).
13. M. Held and R.M. Karp, 1962. A Dynamic Programming Approach to Sequencing Problems, *SIAM Journal on Applied Mathematics 10:* 196–210.
14. S. Lin and B.W. Kernighan, 1973. An Effective Heuristic Algorithm for Traveling-Salesman Problem, *Operations Research 21:* 498–516.
15. O. Martin, S.W. Otto and E.W. Felten, 1990. Large-Step Markov Chains for the TSP, presented at the TSP-Workshop 1990, CRPC, Rice University.
16. M.W. Padberg and G. Rinaldi, 1987. Optimization of a 532-City Symmetric Travelling Salesman Problem by Branch and Cut, *Operations Research Letters 6:* 1–7.
17. M.W. Padberg and G. Rinaldi, 1988. A Branch and Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems, IASI Research Report 247.
18. G. Reinelt, 1989. Fast Heuristics for Large Geometric Travelling Salesman Problems, Report No. 185 Schwerpunktprogramm der Deutschen Forschungsgemeinschaft, Universität Augsburg, Augsburg.
19. T.H.C. Smith and G.L. Thompson, 1977. A LIFO Implicit Enumeration Search Algorithm for the Symmetric Traveling Salesman Problem Using Held and Karp's 1-Tree Relaxation, *Annals of Discrete Mathematics 1:* 479–493.
20. M. Troyon, 1988. Quelques heuristiques et résultats asymptotiques pour trois problèmes d'optimisation combinatoire, Thèse No. 754, Département de Mathématique, Ecole Polytechnique Fédérale de Lausanne.
21. N.L.J. Ulder, E. Pesch, P.J.M. van Laarhoven, H.-J. Bandlet and E.H.L. Aarts, 1990. Improving TSP Exchange Heuristics by Population Genetics, Preprint, Erasmus Universiteit Rotterdam.
22. P.J.M. van Laarhoven, 1988. Theoretical and Computational Aspects of Simulated Annealing, Ph.D. Thesis, Erasmus Universiteit Rotterdam.