TSPLIB File Format

TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types. 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 minimal 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.

Hamiltonian cycle problem (HCP)
Given a graph, test if the graph contains a Hamiltonian cycle or not.

Asymmetric traveling salesman problem (ATSP)
Given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimal 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.

Sequential ordering problem (SOP)
This problem is an asymmetric traveling salesman problem with additional constraints. Given a set of n nodes and distances for each pair of nodes, find a Hamiltonian path from node 1 to node n of minimal length which takes given precedence constraints into account. Each precedence constraint requires that some node i has to be visited before some other node j.

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 minimal 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.)
Except, for the Hamiltonian cycle problems, all problems are defined on a complete graph and, at present, all distances are integer numbers. There is a possibility to require that certain edges appear in the solution of a problem.

1. The file format

Each file consists of a **specification part** and of 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 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 : < *stringy*

Identifies the data file.

1.1.2 TYPE : < *stringy*

Specifies the type of the data. Possible types are

TSP Data for a symmetric traveling salesman problem
ATSP Data for an asymmetric traveling salesman problem
SOP Data for a sequential ordering problem
HCP Hamiltonian cycle problem data
CVEP Capacitated vehicle routing problem data
TOUR A collection of tours

1.1.3 COMMENT : *<stringy*

Additional comments (usually the name of the contributor or creator of the problem instance is given here).

1.1.4 DIMENSION : *<integery*

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. For a TOUR file it is the dimension of the corresponding problem.

1.1.5 CAPACITY : *<integery*

Specifies the truck capacity in a CVRP.

1.1.6 EDGE_WEIGHT_TYPE : *<stringy*

Specifies how the edge weights (or distances) are given. The values are

EXPLICIT Weights are listed explicitly in the corresponding section
EUC_2D Weights are Euclidean distances in 2-D
EUC_3D Weights are Euclidean distances in 3-D

MAX_2D	Weights are maximum distances in 2-D
MAX_3D	Weights are maximum distances in 3-D
MAN_2D	Weights are Manhattan distances in 2-D
MAN_3D	Weights are Manhattan distances in 3-D
CEIL_2D	Weights are Euclidean distances in 2-D rounded up
GEO	Weights are geographical distances
ATT	Special distance function for problems **att48** and **att532**
XEAY1	Special distance function for crystallography problems (Version 1)
XEAY2	Special distance function for crystallography problems (Version 2)
SPECIAL	There is a special distance function documented elsewhere

1.1.7 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_R0W	Upper triangular matrix (row-wise without diagonal entries)
L0WER_R0W	Lower triangular matrix (row-wise without diagonal entries)
UPPER_DIAG_ROW	Upper triangular matrix (row-wise including diagonal entries)
L0WER_DIAG_R0W	Lower triangular matrix (row-wise including diagonal entries)
UPPER.COL	Upper triangular matrix (column-wise without diagonal entries)
L0WER_C0L	Lower triangular matrix (column-wise without diagonal entries)
UPPER_DIAG_COL	Upper triangular matrix (column-wise including diagonal entries)
L0WER_DIAG_C0L	Lower triangular matrix (column-wise including diagonal entries)

1.1.7 EDGEJ3ATA_F0RMAT : <*string*>

Describes the format in which the edges of a graph are given, if the graph is not complete. The values are

EDGE_LIST The graph is given by an edge list

ADJ_LIST The graph is given as an adjacency list

1.1.9 N0DE_C00RD_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

TW0D_C00RDS	Nodes are specified by coordinates in 2-D
THREED.COORDS	Nodes are specified by coordinates in 3-D
N0_C00RDS	The nodes do not have associated coordinates

The default value is N0_C00RDS.

1.1.10 DISPLAY_DATA_TYPE : <*string*>

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

C00RD_DISPLAY	Display is generated from the node coordinates
TWOD_DISPLAY	Explicit coordinates in 2-D are given
NO_DISPLAY	No graphical display is possible

The default value is C00RD_DISPLAY if node coordinates are specified and NO_DISPLAY otherwise.

1.1.11 EOF :
Terminates the input data. This entry is optional.

1.2 The data part
Depending on the choice of specifications some additional data may be required. These data are given in corresponding data sections following the specification part. Each data section begins with 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 N0DE_C00RD.SECTION :

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

<integer> <real> <real>

if N0DE_C00RD_TYPE is TW0D_C00RDS, or

<integer> <real> <real> <real>

if N0DE_C00RD_TYPE is THREED_C00RDS. The integers give the number of the respective nodes. The real numbers give the associated coordinates.

1.2.2 DEP0T_SECTI0N :
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 EDGEJ3ATA_SECTI0N :
Edges of a graph are specified in either of the two formats allowed in the EDGE_DATA_FORMAT entry. If the type is EDGEJLIST, then the edges are given as a sequence of lines of the form
<integer> <integer>
each entry giving the terminal nodes of some edge. The list is terminated by a — 1.
If the type is ADJJLIST, the section consists of a list of adjacency lists for nodes. The adjacency list of a node *x* is specified as
< integer> < integer> ... < integer> — 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.

1.2.5 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.

1.2.6 DISPLAY_DATA_SECTION :

If DISPLAY_DATA_TYPE is TW0DJ3ISPLAY, 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.

1.2.7 T0URJ3ECTI0N :

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.WEIGHT.SECTION :

The edge weights are given in the format specified by the EDGE_WEIGHT_FORMAT entry. At present, all explicit data is integral and is given in one of the (self-explanatory) matrix formats, with implicitly known lengths.

2. The distance functions

For the various choices of EGDE.WEIGHT_TYPE, we now describe the computations of the repsective distances. 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, we round to the nearest integer (in most cases). Below we have used the rounding function **"nint".**

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];
di j = nint( sqrt( xd*xd + yd*yd) );
```

In the 3-dimensional case we have:

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

where **sqrt** is the C square root function.

2.2 Manhattan distance (Li-metric)

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

2-dimensional case:

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

2- 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 );
```

2.3 Maximum distance (Loo-metric)

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

2-dimensional case:

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

3-dimensional case:

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

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 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 different nodes *i* and *j* in kilometers is then computed as follows:

```
ERE = 6378.388;
ql = cos( longitude[i] - longitude[j] );
q2 = cos( latitude[i] - latitude[j] );
q3 = cos( latitude[i] + latitude[j] );
dij = (int) ( EEE * acos( 0.5*((1.0+ql)*q2 - (1.0-ql)*q3) ) + 1.0);
```

The function **"acos"** is the inverse of the 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 (tijcrij) dij = tij + 1;
else dij = tij;
```