TSPLIB 95 Documentation

Release 0.6.1

Robert Grant

Apr 11, 2020

Contents:

CHAPTER 1

TSPLIB 95

TSPLIB 95 is a library for working with TSPLIB 95 files.

- Free software: Apache Software License 2.0
- Documentation: https://tsplib95.readthedocs.io.

For now...

- the documentation is not complete
- only 3.6+ is supported

1.1 Features

- read and use the TSPLIB95 file format like a boss
- easily convert problems into `networkx.Graph` instances
- supports and implements the following `EDGE_WEIGHT_TYPE` s
  - `EXPLICIT`
  - `EUC_2D`
  - `EUC_3D`
  - `MAX_2D`
  - `MAX_3D`
  - `MAN_2D`
  - `MAN_3D`

  - CEIL_2D
  - GEO
  - ATT
  - XRAY1
  - XRAY2
- supports the following EDGE_WEIGHT_FORMAT s
  - FULL_MATRIX
  - UPPER_ROW
  - LOWER_ROW
  - UPPER_DIAG_ROW
  - LOWER_DIAG_ROW
  - UPPER_COL
  - LOWER_COL
  - UPPER_DIAG_COL
  - LOWER_DIAG_COL
- supports SPECIAL FUNCTION edge weights too

It also has a CLI program to print a tabular summary of one or more TSPLIB95 files. No idea why anyone would want that, but there you have it.

1.2 Credits

See TSPLIB for original details, including file format specification, C++ code, and sample problems.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install TSPLIB 95, run this command in your terminal:

```
$ pip install tsplib95
```

This is the preferred method to install TSPLIB 95, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for TSPLIB 95 can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/rhgrant10/tsplib95
```

Or download the tarball:

```
$ curl  -OL https://github.com/rhgrant10/tsplib95/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use TSPLIB 95 in a project:

```
import tsplib95
```

Loading problems and solutions is easy:

```
>>> problem = tsplib95.load_problem('path/to/ulysses16.tsp')
>>> problem
<tsplib95.models.Problem at 0x105030d30>
>>> solution = tsplib95.load_solution('path/to/ulysses16.opt.tour')
>>> solution
<tsplib95.models.Solution at 0x104314d68>
```

Note: **tsplib95** does not ship with any problem files itself. The problems and solutions are standalone files. `load_problem` and `load_solution` take a filesystem path as an argument, not the name of a problem. Feel free to download and use any of the original TSPLIB problem files commonly used and referenced by the community, find others, or write your own. Any file that adheres to the TSPLIB95 file format standards should load without error.

Both have the base attributes, but let's focus on a problem object first:

```
>>> problem.name  # not specified
>>> problem.comment
'Odyssey of Ulysses (Groetschel/Padberg)'
>>> problem.type
'TSP'
>>> problem.dimension
16
```

Problems can be specified in several ways according to the TSPLIB format. Here's how this particular problem is specified:

```
>>> problem.display_data_type
'COORD_DISPLAY'
>>> problem.edge_data_format     # not specified
>>> problem.edge_weight_format   # not specified
>>> problem.edge_weight_type
'GEO'
>>> problem.node_coord_type      # not specified
```

Regardless of how the problem is specified, nodes and edges are accessible in the same way. Nodes and edges are returned as generators since there could be a significant number of them:

```
>>> list(problem.get_nodes())
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
>>> list(problem.get_edges())[:5]
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)]
```

We can find the weight of the edge between nodes 1 and, say, 11, using `wfunc`:

```
>>> problem.wfunc
<function tsplib95.models.Problem._create_distance_function.<locals>.adapter>
>>> problem.wfunc(1, 11)
26
```

If the distance function for the problem is "SPECIAL" you must provide a custom distance function. The function must accept two node coordinates and return the distance between them. Let's create one:

```
>>> import random
>>> import math
>>>
>>> def euclidean_2d_jitter(a, b):
...     x1, y1 = a
...     x2, y2 = b
...     dist = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
...     return dist * random.random() * 2
...
```

Of course, you may want to leverage the existing distance functions:

```
>>> from tsplib95 import distances
>>>
>>> def euclidean_jitter(a, b):
...     dist = distances.euclidean(a, b)  # works for n-dimensions
...     return dist * random.random() * 2
...
```

You can either provide that function at load time or you can also set it on an existing `Problem` instance:

```
>>> problem = tsplib95.load_problem('example.tsp', special=euclidean_2d_jitter)
>>> problem.special = euclidean_jitter
```

Note that setting the special function on a problem that has explicit edge weights has no effect.

You can get a `networkx.Graph` instance from the problem:

```
>>> G = problem.get_graph()
>>> G.nodes
NodeView((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16))
```

And you can trace the tours found in a `Solution`:

```
>>> solution = tsplib95.load_solution('ulysses16.opt.tour')
>>> problem.trace_tours(solution)
[73]
```

Note that it returns a list of tour distances, one for each tour defined in the solution.

CHAPTER 4

API Documentation

4.1 Model classes

class tsplib95.models.**File**(***kwargs*)

Bases: object

Base file format type.

This class isn't meant to be used directly. It contains the common keyword values among all formats. Note that all information is optional. Missing information values are set to None. See the official TSPLIB documentation for more details.

- name - NAME
- comment - COMMENT
- type - TYPE
- dimension - DIMENSION

class tsplib95.models.**Problem**(*special=None*, ***kwargs*)

Bases: *tsplib95.models.File*

A TSPLIB problem file.

Provides a python-friendly way to access the fields of a TSPLIB probem. The fields are mapped as follows:

- name - NAME
- comment - COMMENT
- type - TYPE
- dimension - DIMENSION
- capacity - CAPACITY
- edge_weight_type - EDGE_WEIGHT_TYPE
- edge_weight_format - EDGE_WEIGHT_FORMAT

- `edge_data_format` - EDGE_DATA_FORMAT
- `node_coord_type` - NODE_COORD_TYPE
- `display_data_type` - DISPLAY_DATA_TYPE
- `depots` - DEPOT_SECTION
- `demands` - DEMAND_SECTION
- `node_coords` - NODE_COORD_SECTION
- `edge_weights` - EDGE_WEIGHT_SECTION
- `display_data` - DISPLAY_DATA_SECTION
- `edge_data` - EDGE_DATA_SECTION
- `fixed_edges` - FIXED_EDGES_SECTION

For problems that require a special distance function, you must set the special function in one of two ways:

```
>>> problem = Problem(special=func, ...)  # at creation time
>>> problem.special = func                # on existing problem
```

Special distance functions are ignored for explicit problems but are required for some.

Regardless of problem type or specification, the weight of the edge between two nodes given by index can always be found using `wfunc`. For example, to get the weight of the edge between nodes 13 and 6:

```
>>> problem.wfunc(13, 6)
87
```

The length of a problem is the number of nodes it contains.

get_display(*i*)
Return the display data for node at index *i*, if available.

Parameters **i** (*int*) – node index

Returns display data for node i

get_edges()
Return an iterator over the edges.

Returns edges

Return type iter

get_graph(*normalize=False*)
Return a networkx graph instance representing the problem.

The metadata of the problem is associated with the graph itself. Additional problem information is associated with the nodes and edges. For example:

```
>>> G.graph
{'name': None,
 'comment': '14-Staedte in Burma (Zaw Win)',
 'type': 'TSP',
 'dimension': 14,
 'capacity': None}
>>> G.nodes[1]
{'coord': (16.47, 96.1),
 'display': None,
```

(continues on next page)

(continued from previous page)

```
 'demand': None,
 'is_depot': False}
>>> G.edges[1, 2]
{'weight': 2, 'is_fixed': False}
```

If the graph is not symmetric then a `networkx.DiGraph` is returned. Optionally, the nodes can be renamed to be sequential and zero-indexed.

Parameters **normalize** (*bool*) – rename nodes to be zero-indexed

Returns graph

Return type `networkx.Graph`

get_nodes()
Return an iterator over the nodes.

Returns nodes

Return type iter

is_complete()
Return True if the problem specifies a complete graph.

Return type bool

is_depictable()
Return True if the problem is designed to be depicted.

Return type bool

is_explicit()
Return True if the problem specifies explicit edge weights.

Return type bool

is_full_matrix()
Return True if the problem is specified as a full matrix.

Return type bool

is_special()
Return True if the problem requires a special distance function.

Return type bool

is_symmetric()
Return True if the problem is not asymmetrical.

Note that even if this method returns False there is no guarantee that there are any two nodes with an asymmetrical distance between them.

Return type bool

is_weighted()
Return True if the problem has weighted edges.

Return type bool

special
Special distance function

trace_tours(*solution*)
Calculate the total weights of the tours in the given solution.

Parameters **solution** (*Solution*) – solution with tours to trace

Returns one or more tour weights

Return type list

class tsplib95.models.**Solution**(***kwargs*)

Bases: *tsplib95.models.File*

A TSPLIB solution file containing one or more tours to a problem.

- `name` - NAME
- `comment` - COMMENT
- `type` - TYPE
- `dimension` - DIMENSION
- `tours` - TOUR_SECTION

The length of a solution is the number of tours it contains.

4.2 Matrix weights

class tsplib95.matrix.**FullMatrix**(*numbers*, *size*, *min_index=0*)

Bases: *tsplib95.matrix.Matrix*

A complete square matrix.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

get_index(*i*, *j*)

Return the linear index for the element at (i,j).

Parameters

- **i** (*int*) – row
- **j** (*int*) – column

Returns linear index for element (i,j)

Return type int

class tsplib95.matrix.**HalfMatrix**(*numbers*, *size*, *min_index=0*)

Bases: *tsplib95.matrix.Matrix*

A triangular half-matrix.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

has_diagonal = True

True if the half-matrix includes the diagonal

value_at(*i*, *j*)
Get the element at row *i* and column *j*.

Parameters

- **i** (*int*) – row
- **j** (*int*) – column

Returns value of element at (i,j)

class tsplib95.matrix.**LowerCol**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.UpperRow*

class tsplib95.matrix.**LowerDiagCol**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.UpperDiagRow*

class tsplib95.matrix.**LowerDiagRow**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.HalfMatrix*

Lower-triangular matrix that includes the diagonal.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

get_index(*i*, *j*)
Return the linear index for the element at (i,j).

Parameters

- **i** (*int*) – row
- **j** (*int*) – column

Returns linear index for element (i,j)

Return type int

has_diagonal = True

class tsplib95.matrix.**LowerRow**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.LowerDiagRow*

Lower-triangular matrix that does not include the diagonal.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

has_diagonal = False

class tsplib95.matrix.**Matrix**(*numbers*, *size*, *min_index=0*)
Bases: object

A square matrix created from a list of numbers.

Elements are accessible using matrix notation. Negative indexing is not allowed.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

get_index(*i*, *j*)
Return the linear index for the element at (i,j).

Parameters

- **i** (*int*) – row
- **j** (*int*) – column

Returns linear index for element (i,j)

Return type int

is_valid_row_column(*i*, *j*)
Return True if (i,j) is a row and column within the matrix.

Parameters

- **i** (*int*) – row
- **j** (*int*) – column

Returns whether (i,j) is within the bounds of the matrix

Return type bool

value_at(*i*, *j*)
Get the element at row *i* and column *j*.

Parameters

- **i** (*int*) – row
- **j** (*int*) – column

Returns value of element at (i,j)

class tsplib95.matrix.**UpperCol**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.LowerRow*

class tsplib95.matrix.**UpperDiagCol**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.LowerDiagRow*

class tsplib95.matrix.**UpperDiagRow**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.HalfMatrix*

Upper-triangular matrix that includes the diagonal.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

get_index(*i*, *j*)
Return the linear index for the element at (i,j).

Parameters

- **i** (*int*) – row

- **j** (*int*) – column

Returns linear index for element (i,j)

Return type int

has_diagonal = True

class tsplib95.matrix.**UpperRow**(*numbers*, *size*, *min_index=0*)
Bases: *tsplib95.matrix.UpperDiagRow*

Upper-triangular matrix that does not include the diagonal.

Parameters

- **numbers** (*list*) – the elements of the matrix
- **size** (*int*) – the width (also height) of the matrix
- **min_index** (*int*) – the minimum index

has_diagonal = False

4.3 Distance functions

tsplib95.distances.**euclidean**(*start*, *end*, *round=<function nint>*)
Return the Euclidean distance between start and end.

Parameters

- **start** (*tuple*) – *n*-dimensional coordinate
- **end** (*tuple*) – *n*-dimensional coordinate
- **round** (*callable*) – function to use to round the result

Returns rounded distance

tsplib95.distances.**geographical**(*start*, *end*, *round=<function nint>*, *radius=6378.388*)
Return the geographical distance between start and end.

Parameters

- **start** (*tuple*) – *n*-dimensional coordinate
- **end** (*tuple*) – *n*-dimensional coordinate
- **round** (*callable*) – function to use to round the result
- **radius** (*float*) – the radius of the Earth

Returns rounded distance

tsplib95.distances.**manhattan**(*start*, *end*, *round=<function nint>*)
Return the Manhattan distance between start and end.

Parameters

- **start** (*tuple*) – *n*-dimensional coordinate
- **end** (*tuple*) – *n*-dimensional coordinate
- **round** (*callable*) – function to use to round the result

Returns rounded distance

tsplib95.distances.**maximum**(*start*, *end*, *round=<function nint>*)
Return the Maximum distance between start and end.

Parameters

- **start** (*tuple*) – *n*-dimensional coordinate
- **end** (*tuple*) – *n*-dimensional coordinate
- **round** (*callable*) – function to use to round the result

Returns rounded distance

tsplib95.distances.**pseudo_euclidean**(*start*, *end*, *round=<function nint>*)
Return the pseudo-Euclidean distance between start and end.

Parameters

- **start** (*tuple*) – *n*-dimensional coordinate
- **end** (*tuple*) – *n*-dimensional coordinate
- **round** (*callable*) – function to use to round the result

Returns rounded distance

tsplib95.distances.**xray**(*start*, *end*, *sx=1*, *sy=1*, *sz=1*, *round=<function icost>*)
Return x-ray crystallography distance.

Parameters

- **start** (*tuple*) – 3-dimensional coordinate
- **end** (*tuple*) – 3-dimensional coordinate
- **sx** (*float*) – x motor speed
- **sy** (*float*) – y motor speed
- **sz** (*float*) – z motor speed

Returns distance

4.4 Utilities

tsplib95.utils.**load_problem**(*filepath*, *special=None*)
Load a problem at the given filepath.

Parameters

- **filepath** (*str*) – path to a TSPLIB problem file
- **special** (*callable*) – special/custom distance function

Returns problem instance

Return type Problem

tsplib95.utils.**load_problem_fromstring**(*text*, *special=None*)
Load a problem from raw text.

Parameters

- **text** (*str*) – text of a TSPLIB problem
- **special** (*callable*) – special/custom distance function

Returns problem instance

Return type Problem

tsplib95.utils.**load_solution**(*filepath*)
Load a solution at the given filepath.

Parameters **filepath** (*str*) – path to a TSPLIB solution file

Returns solution instance

Return type Solution

tsplib95.utils.**load_solution_fromstring**(*text*)
Load a solution from raw text.

Parameters **text** (*str*) – text of a TSPLIB solution

Returns solution instance

Return type Solution

tsplib95.utils.**load_unknown**(*filepath*)
Load any TSPLIB file.

This is particularly useful when you do not know in advance whether the file contains a problem or a solution.

Parameters **filepath** (*str*) – path to a TSPLIB problem file

Returns either a problem or solution instance

tsplib95.utils.**load_unknown_fromstring**(*text*)
Load any problem/solution from raw text.

This is particularly useful when you do not know in advance whether the file contains a problem or a solution.

Parameters **text** (*str*) – text of a TSPLIB problem/solution

Returns either a problem or solution instance

tsplib95.utils.**nint**(*x*)
Round a value to an integer.

Parameters **x** (*float*) – original value

Returns rounded integer

Return type int

tsplib95.utils.**parse_degrees**(*coord*)
Parse an encoded geocoordinate value into real degrees.

Parameters **coord** (*float*) – encoded geocoordinate value

Returns real degrees

Return type float

4.5 Exceptions

class tsplib95.parser.**ParsingError**(**args*, ***kwargs*)
Error raised when a given file cannot be parsed.

This indicates the file does not conform to the standard TSPLIB95 file format.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/rhgrant10/tsplib95/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

TSPLIB 95 could always use more documentation, whether as part of the official TSPLIB 95 docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/rhgrant10/tsplib95/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *tsplib95* for local development.

1. Fork the *tsplib95* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/tsplib95.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv tsplib95
$ cd tsplib95/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 tsplib95 tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/rhgrant10/tsplib95/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_tsplib95
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Robert Grant <rhgrant10@gmail.com>

6.2 Contributors

- Michael Ritter <michael.ritter^P@tum.de>

CHAPTER 7

History

7.1 0.6.1 (2020-01-04)

- Fix bug that caused the parser to ignore the first line of a file

7.2 0.6.0 (2019-10-19)

- Changes to the conversion into a `networkx.Graph`:
  - Depot, demand, and fixed edge data have been removed from graph metadata. Depot and demand data is now associated with individual nodes like fixed edge data was (and still is).
  - Add a `normalized` parameter to allow nodes to be renamed as zero-index integers when obtaining a `networkx.Graph`.
- Depots, demands, node coordinates, and display data fields now default to empty containers rather than `None`.
- Fixed twine/PyPI warning about long description mime type

7.3 0.5.0 (2019-10-02)

- New loaders that take just the text - no file necessary!
- Invalid keywords now result in a `ParsingError`
- Update the CLI to catch and gracefully handle `ParsingError`
- Fixed a bug when trying to amend an exception with line information

7.4 0.4.0 (2019-09-21)

- All expected parsing errors are now raised as `ParsingError` rather than the base `Exception` type.
- Fix name of distance paramter to distances.geographical. Previously it was "diameter" but was used as a radius. It is now "radius".
- Relax restriction on networkx version (now `~=2.1`)
- Add documentation for each problem field
- Other minor documentation changes
- Add offical 3.7 support
- Add missing history entry for v0.3.3
- Remove some dead code

7.5 0.3.3 (2019-03-24)

- Fix parsing bug for key-value lines whose value itself contains colons

7.6 0.3.2 (2018-10-07)

- Fix bug in `Problem.is_complete` that produced a `TypeError` when run
- Fix bug in `Problem.is_depictable` that produced a `TypeError` when run
- Fix bug in `Problem.get_display` that produced an `AttributeError` when run
- Added some unit tests for the `Problem` class
- Added some unit tests for the `parser` module

7.7 0.3.1 (2018-10-03)

- Fix bug in `Problem.is_weighted` that caused problems with defined nodes coords to use the unit distance function

7.8 0.3.0 (2018-08-12)

- Added XRAY1 and XRAY2 implementations
- Simplified some of the matrix code

7.9 0.2.0 (2018-08-12)

- Implement column-wise matrices
- Add a utiltiy for loading an unknown file

- Fix bug in the ATT distance function
- Update the CLI to use the models
- Document a bunch-o-stuff
- Switch to RTD sphinx theme
- Move most utilties into utils

7.10 0.1.0 (2018-08-12)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index