TSPLIB 95 Documentation

Release 0.7.1

Robert Grant

May 08, 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.

1.1 Features

- **read** and **write** TSPLIB95 file format like a boss
- easily **convert** problems into `networkx.Graph` instances
- supports **all** fields in the original standard
- allows completely **custom** field and problem declarations

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 nonetheless.

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
```

Note: **tsplib95** does not officially ship with any problem files itself. The problems and solutions are standalone files. 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.

3.1 Loading problems

Problems can be loaded from files, read from file pointers, or parsed from strings.

- *tsplib95.load()*: loads from a filepath
- *tsplib95.parse()*: parses directly from a string
- *tsplib95.read()*: reads from a file-like object

3.1.1 From a filepath

For the simple case of a file on disk, pass the filepath to `tsplib95.load()`:

```
>>> import tsplib95
>>> problem = tsplib95.load('archives/problems/tsp/bay29.tsp')
```

3.1.2 From a string

For cases where the problem isn't stored as a file on disk, just pass the text directly to `tsplib95.parse()`:

```
>>> import tsplib95
>>> with open('archives/problems/tsp/gr17.tsp') as f:
...     text = f.read()
...
>>> problem = tsplib95.parse(text)
```

3.1.3 From a file-like object

If you already have an open file-like object, just pass it to `tsplib95.read()`:

```
>>> import tsplib95
>>> with open('archives/problems/tsp/gr17.tsp') as f:
...     problem = tsplib95.read(f)
...
```

3.1.4 SPECIAL functions

What's a SPECIAL function?

Some problems involve using a custom function for calcating edge weights. Such custom weight functions are called "special" functions because their details are not defined by TSPLIB95 standard.

Problems that use a special function have the following characteristics:

- EDGE_WEIGHT_FORMAT is "FUNCTION"
- EDGE_WEIGHT_TYPE is "SPECIAL"

In *tsplib95*, a special function must accept two node coordinates and return the weight, distance, or cost of the edge between them:

```
from typing import Union, Sequence

def special(start: Sequence, end: Sequence) -> Union[float,int]:
    pass
```

How to use

All loaders (*`load()`*, *`read()`*, and *`parse()`*) accept a custom weight function via the keyword parameter *special*.

```
>>> import tsplib95
>>> from myapp import get_distance
>>> problem = tsplib95.load('assets/tsp/routes-150.tsp',
...                         special=get_distance)
```

Special functions can also be set on an existing problem instance:

```
>>> import tsplib95
>>> from myapp import get_distance
>>> problem = tsplib95.load('assets/tsp/routes-150.tsp')
>>> problem.special = get_distance
```

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

An example

Let's assume our app has a helper function capable of using the Google Maps API to fetch the driving distance between two geocoordinates. We can use that to create a special function and use it as the distance function in a TSPLIB95 problem.

Let's also assume our hypothetical helper function accepts a list of waypoints as dictionaries with "lat" and "lng" keys, but our problem specifies geocoordinates as a simple tuple of latitude and longitude values.

Our special function will need to convert the tuples into the expected dictionaries and use the helper to calculate the driving distance.

```
>>> from myapp import helpers

>>> def waypoint(coordinates):
...     return {'lat': coordinates[0], 'lng': coordinates[1]}

>>> def driving_distance(start, end):
...     """Special distance function for driving distance."""
...     waypoints = [waypoint(start), waypoint(end)]
...     kilometers = helpers.total_distance(waypoints, method='driving')
...     return kilometers
...
```

Now we can simply supply the *tsplib95.load* method with our special function:

```
>>> import tsplib95
>>> problem = tsplib95.load('my-hometown.tsp', special=driving_distance)
```

As with any problem, we can find the weight of any edge using the *get_weight()* method. See the *Distances* section for more details.

Note: The example above demonstrates the flexibility of the special function, but depending on the specific implementation details of the helper function, it could be too fragile.

There is no exception handling around the use of the special function so it is advisable to handle any exceptions. Depending on the use case, it may also be wise to limit calls to it by wrapping it in a debounce function or backing it with a cache.

3.2 Saving problems

The `Problem` class also two convenience methods for output to files:

- `Problem.save`: saves to a filepath
- `Problem.write`: writes to a file-like object

```
>>> problem.save('path/to/file.name')

>>> with open('path/to/other.name', 'w') as f:
...     problem.write(f)
...
```

3.3 Rendering problems

Problems can be rendered back into text using the `Problem.render` method:

```
>>> print(problem.render())
NAME: gr17
COMMENT: 17-city problem (Groetschel)
TYPE: TSP
DIMENSION: 17
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: LOWER_DIAG_ROW
EDGE_WEIGHT_SECTION:
0 633 0 257 390 0 91 661 228 0 412 227
169 383 0 150 488 112 120 267 0 80 572 196
77 351 63 0 134 530 154 105 309 34 29 0
259 555 372 175 338 264 232 249 0 505 289 262
476 196 360 444 402 495 0 353 282 110 324 61
208 292 250 352 154 0 324 638 437 240 421 329
297 314 95 578 435 0 70 567 191 27 346 83
47 68 189 439 287 254 0 211 466 74 182 243
105 150 108 326 336 184 391 145 0 268 420 53
239 199 123 207 165 383 240 140 448 202 57 0
246 745 472 237 528 364 332 349 202 685 542 157
289 426 483 0 121 518 142 84 297 35 29 36
236 390 238 301 55 96 153 336 0
EOF
```

Note this is equivalent to casting the problem to a string:

```
>>> assert str(problem) == problem.render()
```

3.4 Working with problems

Note: In general, familiarity with the original file format standard will translate well. Please refer to it for an better understanding of the underlying TSPLIB95 file format.

3.4.1 Accessing Values

In general, the name of a field is its keyword converted to lowercase. For example, "NAME" is `name` and "EDGE_WEIGHT_FORMAT" is `edge_weight_format`, *etc.*:

```
>>> problem.name  # NAME
'gr17'
>>> problem.edge_weight_format  # EDGE_WEIGHT_FORMAT
'LOWER_DIAG_ROW'
```

However, field names do *not* have the "_SECTION" suffix of some keywords:

```
>>> problem.edge_weights  # not EDGE_WEIGHT_SECTION
[[0, 633, 0, 257, 390, 0, 91, 661, 228, 0, 412, 227],
 [169, 383, 0, 150, 488, 112, 120, 267, 0, 80, 572, 196],
```

(continues on next page)

(continued from previous page)

```
 [77, 351, 63, 0, 134, 530, 154, 105, 309, 34, 29, 0],
 [259, 555, 372, 175, 338, 264, 232, 249, 0, 505, 289, 262],
 [476, 196, 360, 444, 402, 495, 0, 353, 282, 110, 324, 61],
 [208, 292, 250, 352, 154, 0, 324, 638, 437, 240, 421, 329],
 [297, 314, 95, 578, 435, 0, 70, 567, 191, 27, 346, 83],
 [47, 68, 189, 439, 287, 254, 0, 211, 466, 74, 182, 243],
 [105, 150, 108, 326, 336, 184, 391, 145, 0, 268, 420, 53],
 [239, 199, 123, 207, 165, 383, 240, 140, 448, 202, 57, 0],
 [246, 745, 472, 237, 528, 364, 332, 349, 202, 685, 542, 157],
 [289, 426, 483, 0, 121, 518, 142, 84, 297, 35, 29, 36],
 [236, 390, 238, 301, 55, 96, 153, 336, 0]]
```

All values are available mapped either by keyword or name:

```
>>> problem.as_name_dict()
{'name': 'gr17',
 'comment': '17-city problem (Groetschel)',
 'type': 'TSP',
 'dimension': 17,
 'capacity': 0,
 'node_coord_type': None,
 'edge_weight_type': 'EXPLICIT',
 'display_data_type': None,
 'edge_weight_format': 'LOWER_DIAG_ROW',
 'edge_data_format': None,
 'node_coords': {},
 'edge_data': {},
 'edge_weights': [[0, 633, 0, 257, 390, 0, 91, 661, 228, 0, 412, 227],
  [169, 383, 0, 150, 488, 112, 120, 267, 0, 80, 572, 196],
  [77, 351, 63, 0, 134, 530, 154, 105, 309, 34, 29, 0],
  [259, 555, 372, 175, 338, 264, 232, 249, 0, 505, 289, 262],
  [476, 196, 360, 444, 402, 495, 0, 353, 282, 110, 324, 61],
  [208, 292, 250, 352, 154, 0, 324, 638, 437, 240, 421, 329],
  [297, 314, 95, 578, 435, 0, 70, 567, 191, 27, 346, 83],
  [47, 68, 189, 439, 287, 254, 0, 211, 466, 74, 182, 243],
  [105, 150, 108, 326, 336, 184, 391, 145, 0, 268, 420, 53],
  [239, 199, 123, 207, 165, 383, 240, 140, 448, 202, 57, 0],
  [246, 745, 472, 237, 528, 364, 332, 349, 202, 685, 542, 157],
  [289, 426, 483, 0, 121, 518, 142, 84, 297, 35, 29, 36],
  [236, 390, 238, 301, 55, 96, 153, 336, 0]],
 'display_data': {},
 'fixed_edges': [],
 'depots': [],
 'demands': {},
 'tours': []}

>>> problem.as_keyword_dict()
{'NAME': 'gr17',
 'COMMENT': '17-city problem (Groetschel)',
 'TYPE': 'TSP',
 'DIMENSION': 17,
 'CAPACITY': 0,
 'NODE_COORD_TYPE': None,
 'EDGE_WEIGHT_TYPE': 'EXPLICIT',
 'DISPLAY_DATA_TYPE': None,
 'EDGE_WEIGHT_FORMAT': 'LOWER_DIAG_ROW',
 'EDGE_DATA_FORMAT': None,
```

(continues on next page)

(continued from previous page)

```
 'NODE_COORD_SECTION': {},
 'EDGE_DATA_SECTION': {},
 'EDGE_WEIGHT_SECTION': [[0, 633, 0, 257, 390, 0, 91, 661, 228, 0, 412, 227],
  [169, 383, 0, 150, 488, 112, 120, 267, 0, 80, 572, 196],
  [77, 351, 63, 0, 134, 530, 154, 105, 309, 34, 29, 0],
  [259, 555, 372, 175, 338, 264, 232, 249, 0, 505, 289, 262],
  [476, 196, 360, 444, 402, 495, 0, 353, 282, 110, 324, 61],
  [208, 292, 250, 352, 154, 0, 324, 638, 437, 240, 421, 329],
  [297, 314, 95, 578, 435, 0, 70, 567, 191, 27, 346, 83],
  [47, 68, 189, 439, 287, 254, 0, 211, 466, 74, 182, 243],
  [105, 150, 108, 326, 336, 184, 391, 145, 0, 268, 420, 53],
  [239, 199, 123, 207, 165, 383, 240, 140, 448, 202, 57, 0],
  [246, 745, 472, 237, 528, 364, 332, 349, 202, 685, 542, 157],
  [289, 426, 483, 0, 121, 518, 142, 84, 297, 35, 29, 36],
  [236, 390, 238, 301, 55, 96, 153, 336, 0]],
 'DISPLAY_DATA_SECTION': {},
 'FIXED_EDGES_SECTION': [],
 'DEPOT_SECTION': [],
 'DEMAND_SECTION': {},
 'TOUR_SECTION': []}
```

3.4.2 Nodes and edges

To help users avoid the complexity in listing the nodes and edges reliably in all cases, there exists the *get_nodes()* and *get_edges()* methods.

```
>>> list(problem.get_nodes())
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

>>> len(list(problem.get_edges()))  # I'll spare you the full listing :P
289

>>> list(problem.get_edges())[0]
(0, 0)
```

3.4.3 Distances

Regardless of whether the problem is explicit or function, the distance between two nodes can always be found by passing their indicies to `get_weight()`.

```
>>> problem = tsplib95.load('archives/problems/vrp/eil22.vrp')
>>> list(problem.get_nodes())
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
>>> problem.node_coords[3]
[159, 261]
>>> problem.node_coords[8]
[161, 242]
>>> problem.edge_weight_type
'EUC_2D'
>>> edge = 3, 8
>>> weight = problem.get_weight(*edge)
>>> print(f'The driving distance from node {edge[0]} to node {edge[1]} is {weight}.')
The distance between node 3 and node 8 is 19.
```

`tsplib95` has built-in support for all function types, including special functions. See *SPECIAL functions* for more information.

3.4.4 Boolean methods

Problems contain a set of functions that report "emergent" boolean information about the problem given its data.

- *is_explicit()*
- *is_full_matrix()*
- *is_weighted()*
- *is_special()*
- *is_complete()*
- *is_symmetric()*
- *is_depictable()*

3.4.5 Extra attributes

There can be no extra or unknown keywords in a problem. Typically, this results in a failure to parse the field preceding it, but not necessarily (for example, if the presence of the keyword in the value of the preceding field is valid then there will be no error).

However, not all fields defined on a `Problem` must be present in a problem. In such a case, requesting the value on the problem instance returns the default value for the field. Fields are not rendered or written to file unless their value has been set.

3.4.6 Tracing tours

Some TSPLIB95 files have a TOURS field that lists one or more tours. Often, the tour(s) are in a separate `.opt.tour` file. *StandardProblem* has a TOURS field, which means it can parse these `.opt.tour` files as well:

```
>>> opt = tsplib95.load('archives/solutions/tour/gr666.opt.tour')
>>> opt.type
'TOUR'
>>> len(opt.tours)
1
>>> len(opt.tours[0])
666
```

We have 1 tour to trace. We can simply pass the list of tours to *trace_tours()*:

```
>>> problem = tsplib95.load('archives/problems/tsp/gr666.tsp')
>>> >>> problem.trace_tours(opt.tours)
[294358]
```

Note: Note that it returned a list of tour weights, one for each tour given.

For testing purposes, there is also *trace_canonical_tour()*, which uses the nodes in definition order as a tour and returns the total weight:

```
>>> weight = problem.trace_canonical_tour()
```

3.5 Converting problems

get_graph() creates a `networkx.Graph` instance from the problem data:

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

The graph, nodes, and edges of the object contain as much accessory information as is present in the problem instance:

```
>>> G.graph
{'name': 'gr17',
'comment': '17-city problem (Groetschel)',
'type': 'TSP',
'dimension': 17,
'capacity': 0}
>>> G.nodes[0]
{'coord': None, 'display': None, 'demand': None, 'is_depot': False}
>>> G.edges[0, 1]
{'weight': 633, 'is_fixed': False}
```

CHAPTER 4

API Documentation

4.1 Loaders

The loaders are responsible for loading from a filepath, reading a file-like object, or parsing a string.

tsplib95.loaders.**load**(*filepath*, *problem_class=None*, *special=None*)
Load a problem at the given filepath.

Parameters

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

Returns problem instance

Return type Problem

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

param str filepath path to a TSPLIB problem file

param callable special special/custom distance function

return problem instance

rtype Problem

Deprecated since version 7.0.0: Will be removed in newer versions. Use *tsplib95.load* instead.

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

param str text text of a TSPLIB problem

param callable special special/custom distance function

return problem instance

rtype Problem

Deprecated since version 7.0.0: Will be removed in newer versions. Use *tsplib95.parse* instead.

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

param str filepath path to a TSPLIB solution file

return solution instance

rtype Solution

Deprecated since version 7.0.0: Will be removed in newer versions. Use *tsplib95.load* instead.

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

param str text text of a TSPLIB solution

return solution instance

rtype Solution

Deprecated since version 7.0.0: Will be removed in newer versions. Use *tsplib95.parse* instead.

tsplib95.loaders.**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.

param str filepath path to a TSPLIB problem file

return either a problem or solution instance

Deprecated since version 7.0.0: Will be removed in newer versions. Use *tsplib95.load* instead.

tsplib95.loaders.**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.

param str text text of a TSPLIB problem/solution

return either a problem or solution instance

Deprecated since version 7.0.0: Will be removed in newer versions. Use *tsplib95.parse* instead.

tsplib95.loaders.**parse**(*text*, *problem_class=None*, *special=None*)
Load a problem from raw text.

Parameters

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

Returns problem instance

Return type Problem

tsplib95.loaders.**read**(*f*, *problem_class=None*, *special=None*)
Read a problem from a file-like object.

Parameters

- **f** (*file*) – file-like object
- **problem_class** (*type*) – special/custom problem class
- **special** (*callable*) – special/custom distance function

Returns problem instance

Return type Problem

4.2 Problems

Problem classes define what fields can be present and how they are converted to and from text.

class tsplib95.models.**Problem**(***data*)

Bases: object

Base class for all problems.

Parameters **data** – name-value data

as_dict(*by_keyword=False*)

Return the problem data as a dictionary.

Parameters **by_keyword** (*bool*) – use keywords (True) or names (False) or keys

Returns problem data

Return type dict

as_keyword_dict()

Return the problem data as a dictionary by field keyword.

Returns problem data

Return type dict

as_name_dict()

Return the problem data as a dictionary by field name.

Returns problem data

Return type dict

classmethod load(*filepath*, ***options*)

Load a problem instance from a text file.

Any keyword options are passed to the class constructor. If a keyword argument has the same name as a field then they will collide and cause an error.

Parameters

- **filepath** (*str*) – path to a problem file
- **options** – any keyword arguments to pass to the constructor

Returns problem instance

Return type *Problem*

classmethod parse(*text*, ***options*)

Parse text into a problem instance.

Any keyword options are passed to the class constructor. If a keyword argument has the same name as a field then they will collide and cause an error.

Parameters

- **text** (*str*) – problem text
- **options** – any keyword arguments to pass to the constructor

Returns problem instance

Return type *Problem*

classmethod read(*fp*, ***options*)

Read a problem instance from a file-like object.

Any keyword options are passed to the class constructor. If a keyword argument has the same name as a field then they will collide and cause an error.

Parameters

- **fp** (*str*) – a file-like object
- **options** – any keyword arguments to pass to the constructor

Returns problem instance

Return type *Problem*

class tsplib95.models.**StandardProblem**(*special=None*, ***data*)

Bases: *tsplib95.models.Problem*

Standard problem as outlined in the original TSLIB95 documentation.

The available fields and their keywords are:

- `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
- `tours` - TOUR_SECTION

For SPECIAL FUNCTION problems, the special function must accept a start and an end node and return the weight, distance, or cost of the edge that joins them. It can be provided at construction time or simply set on an existing object using the `special` attribute.

Parameters

- **special** (*callable*) – special function for distance
- **data** – name-value data

get_display(*i*)

Return the display data for node at index *i*.

If the problem is not depictable, None is returned instead.

Parameters **i** (*int*) – node index

Returns display data for node i

get_edges()

Return an iterator over the edges.

This method provides a single way to obtain the edges of a problem regardless of how it is specified. If the EDGE_DATA_FORMAT is not set and the nodes are undefined, then the edges are also undefined.

Returns edges

Return type iter

Raises **ValueError** – if the nodes and therefore the edges are undefined

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

If the graph is asymmetric 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.

This method provides a single way to obtain the nodes of a problem regardless of how it is specified. However, if the nodes are not specified, the EDGE_DATA_FORMAT is not set, and DIMENSION has no value, then nodes are undefined.

Returns nodes

Return type iter

Raises **ValueError** – if the nodes are undefined

get_weight(*start*, *end*)
Return the weight of the edge between start and end.

This method provides a single way to obtain edge weights regardless of whether the problem uses an explicit matrix or a distance function.

Parameters

- **start** (*int*) – starting node index
- **end** (*int*) – ending node index

Returns weight of the edge between start and end

Return type float

is_complete()
Check whether the problem specifies a complete graph.

Returns True if the problem specifies a complete graph

Return type bool

is_depictable()
Check whether the problem can be depicted.

A problem is depictable if it has display data or has node coordinates and does not specify NO_DISPLAY.

Returns True if the problem can be depicted

Return type bool

is_explicit()
Check whether the problem specifies edge weights explicitly.

Returns True if the problem specifies edge weights explicitly

Return type bool

is_full_matrix()
Check whether the problem is specified as a full matrix.

Returns True if the problem is specified as a full matrix

Return type bool

is_special()
Check whether the problem is special.

SPECIAL problems require a special distance function.

Returns True if the problem requires a special distance function

Return type bool

is_symmetric()
Check whether the problem is symmetrical.

> **Warning:** Although a result of `True` guarantees symmetry, a value of `False` merely indicates the *possibliity* for asymmetry. Avoid using `not problem.is_symmetric()` when possible.

Returns True if the problem is symmetrical

Return type bool

is_weighted()
Check whether the problem has weighted edges.

A problem is considered unweighted if neither the EDGE_WEIGHT_FORMAT nor the EDGE_WEIGHT_TYPE are defined.

Returns True if the problem is weighted

Return type bool

special
Special distance function.

Special/custom distance functions must accept two coordinates of appropriate dimension and return the distance between them.

trace_canonical_tour()
Return the weight of the canonical tour.

The "canonical tour" uses the nodes in order. This method is present primarily for testing and purposes.

Returns weight of the canonical tour

Return type float

trace_tours(*tours*)
Return the weights of the given tours.

Each tour is a list of node indices. The weights returned are the sum of the individual weights of the edges in each tour including the final edge back to the starting node.

The list of weights returned parallels the list of tours given so that `weights[i]` corresponds to `tours[i]`:

```
weights = p.trace_tours(tours)
```

Parameters **tours** (*list*) – one or more lists of node indices

Returns one weight for each given tour

Return type list

4.3 Fields

class tsplib95.fields.**Field**(*keyword*, *, *default=None*)
Bases: `object`

Contains base functionality for all fields.

The default value can be a callable, in which case it is invoked for each call to *get_default_value()*. The default can be set on an instance or as a class attribute, but the class attribute is only checked when the field is initially created.

Parameters

- **keyword** (*str*) – keyword (typically all caps)
- **default** – a default value or callable that will return a default

get_default_value()
Return the default value.

Callables are called for a default value to return each time.

Returns the default value

Return type Any

parse(*text*)
Convert text into a value.

This must be implemented in a subclass.

Parameters **text** (*str*) –

Returns a value

render(*value*)
Convert a value into text.

This must be implemented in a subclass.

Parameters **value** – a value

Returns text

Return type str

validate(*value*)
Validate a value.

Raise an exception if the value fails validation.

The default implementation does nothing.

Parameters **value** – a value

Raises **Exception** – if the value does not pass validation

class tsplib95.fields.**TransformerField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.Field*

Field that delegates to a *Transformer*.

Parameters

- **keyword** (*str*) – keyword
- **transformer** (*callable*) – transformer to use

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

parse(*text*)
Parse the text into a value using the transformer.

Parameters **text** (*str*) – text to parse

Returns value

render(*value*)
Render the value into text using the transformer.

Parameters **text** (*str*) – value to render

Returns text

validate(*value*)
Validate the value using the transformer.

Parameters **text** (*str*) – value to validate

class tsplib95.fields.**StringField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Simple string field.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

class tsplib95.fields.**IntegerField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Simple integer field.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

class tsplib95.fields.**FloatField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Simple float field.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

class tsplib95.fields.**NumberField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Number field, supporting ints and floats.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

class tsplib95.fields.**IndexedCoordinatesField**(**args*, *dimensions=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for coordinates by index.

When given, `dimensions` stipulates the possible valid dimensionalities for the coordinates. For exapmle, `dimensions=(2, 3)` indicates the coordinates are either all 2d or all 3d, whereas `dimensions=2` indicates all coordinates must be 2d. The check is only enforced during validation.

Parameters **dimensions** – one or more valid dimensionalities

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of `builtins.dict`

validate(*value*)
Validate the value using the transformer.

Parameters **text** (*str*) – value to validate

class tsplib95.fields.**AdjacencyListField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for an adjancency list.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of `builtins.dict`

class tsplib95.fields.**EdgeListField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for a list of edges.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of `builtins.list`

class tsplib95.fields.**MatrixField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for a matrix of numbers.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of `builtins.list`

class tsplib95.fields.**EdgeDataField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for edge data.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of builtins.dict

class tsplib95.fields.**DepotsField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for depots.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of builtins.list

class tsplib95.fields.**DemandsField**(*keyword*, *, *transformer=None*, ***kwargs*)
Bases: *tsplib95.fields.TransformerField*

Field for demands.

classmethod build_transformer()
Construct an appropriate transformer for the field.

Returns transformer

Return type *Transformer*

default
alias of builtins.dict

class tsplib95.fields.**ToursField**(**args*, *require_terminal=True*)
Bases: *tsplib95.fields.Field*

Field for one or more tours.

default
alias of builtins.list

parse(*text*)
Parse the text into a list of tours.

Parameters **text** (*str*) – text to parse

Returns tours

Return type list

render(*tours*)
Render the tours as text.

Parameters **tours** (*list*) – tours to render

Returns rendered text

Return type str

4.4 Transformers

class tsplib95.transformers.**Transformer**
Bases: object

Reusable transformer between text and data.

parse(*text*)
Return the value of the text.

Parameters **text** (*str*) – the text

Returns the value

Raises ***ParsingError*** – if the text cannot be parsed into a value

render(*value*)
Return the text for the value.

Parameters **value** (*str*) – the value

Returns the text

validate(*value*)
Validate the value.

Parameters **value** – the value

class tsplib95.transformers.**FuncT**(*, *func*)
Bases: *tsplib95.transformers.Transformer*

Transformer that simply wraps a parsing function.

The parsing function must accept a single positional argument for the text to parse. It must parse and return the value.

Values are rendered back into values using the builtin `str()`, so it's generally best to use this for primitives.

Parameters **func** (*callable*) – parsing function

parse(*text*)
Return the value of the text.

Parameters **text** (*str*) – the text

Returns the value

Raises ***ParsingError*** – if the text cannot be parsed into a value

class tsplib95.transformers.**NumberT**
Bases: *tsplib95.transformers.Transformer*

Transformer for any number, int or float.

parse(*text*)
Return the value of the text.

Parameters **text** (*str*) – the text

Returns the value

Raises ***ParsingError*** – if the text cannot be parsed into a value

class tsplib95.transformers.**ContainerT**(*, *value=None*, *sep=None*, *terminal=None*, *terminal_required=True*, *size=None*, *filter_empty=True*)

Bases: *tsplib95.transformers.Transformer*

Transformer that acts as a generic container.

Parameters

- **value** (*Transformer*) – transformer for each item
- **sep** (*str*) – separator between items
- **terminal** (*str*) – text that marks the end
- **terminal_required** (*bool*) – whether the terminal is required
- **size** (*int*) – required number of items
- **filter_empty** (*bool*) – filter out empty items (zero-length/blank)

join_items(*items*)

Join zero or more items into a single text.

Parameters **items** (*list*) – items to join

Returns joined text

Return type str

pack(*items*)

Pack the given items into a container.

Parameters **items** (*list*) – items to pack

Returns container with items in it

parse(*text*)

Parse the text into a container of items.

Parameters **text** (*str*) – the text to parse

Returns container

parse_item(*text*)

Parse the text into a single item.

Parameters **text** (*str*) – the text to parse

Returns container

render(*container*)

Render the container into text.

Parameters **container** – container to render

Returns text

render_item(*item*)

Render the item into text.

Parameters **item** – item to render

Returns text

split_items(*text*)

Split the text into multiple items.

Parameters **text** (*str*) – text to split

Returns muliple items

Return type list

unpack(*container*)
Unpack items from the container.

Parameters **container** – container with items in it

Returns items from container

Return type list

class tsplib95.transformers.**ListT**(*, *value=None*, *sep=None*, *terminal=None*, *terminal_required=True*, *size=None*, *filter_empty=True*)
Bases: *tsplib95.transformers.ContainerT*

Transformer for a list of items.

pack(*items*)
Pack the given items into a container.

Parameters **items** (*list*) – items to pack

Returns container with items in it

unpack(*container*)
Unpack items from the container.

Parameters **container** – container with items in it

Returns items from container

Return type list

class tsplib95.transformers.**MapT**(*, *kv_sep=None*, *key=None*, ***kwargs*)
Bases: *tsplib95.transformers.ContainerT*

Transformer for a key-value mapping of items.

Parameters

- **kv_sep** (*str*) – separator between keys and values
- **key** (*Transformer*) – transformer for the keys

pack(*items*)
Pack the given items into a container.

Parameters **items** (*list*) – items to pack

Returns container with items in it

parse_item(*text*)
Parse the text into a single item.

Parameters **text** (*str*) – the text to parse

Returns container

parse_key(*text*)
Parse the text into a key.

Parameters **text** (*str*) – the text to parse

Returns key

parse_value(*text*)
Parse the text into a value.

Parameters **text** (*str*) – the text to parse

Returns value

render_item(*item*)
Render the item into text.

Parameters **item** – item to render

Returns text

render_key(*key*)
Render the key into text.

Parameters **key** – the key to render

Returns text

Return type str

render_value(*value*)
Render the value into text.

Parameters **value** – the value to render

Returns text

Return type str

unpack(*container*)
Unpack items from the container.

Parameters **container** – container with items in it

Returns items from container

Return type list

class tsplib95.transformers.**UnionT**(**tfs*, ***kwargs*)
Bases: *tsplib95.transformers.Transformer*

parse(*text*)
Return the value of the text.

Parameters **text** (*str*) – the text

Returns the value

Raises ***ParsingError*** – if the text cannot be parsed into a value

render(*value*)
Return the text for the value.

Parameters **value** (*str*) – the value

Returns the text

4.5 Matrices

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

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

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

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

4.6 Distances

tsplib95.distances.**TYPES = {'ATT': <function pseudo_euclidean>, 'CEIL_2D': functools.partia**
Map of distance function types to distance functions

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

This is capable of performing distance calculations for EUC_2D and EUC_3D problems. If `round=math.ceil` is passed, this is suitable for CEIL_2D problems as well.

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

This is capable of performing distance calculations for MAN_2D and MAN_3D problems.

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.

This is capable of performing distance calculations for MAX_2D and MAX_3D problems.

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.

This is capable of performing distance calculations for GEO problems.

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

This is capable of performing distance calculations for ATT problems.

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=1.0*, *sy=1.0*, *sz=1.0*, *round=<function nint>*)
Return x-ray crystallography distance.

This is capable of performing distance calculations for xray crystallography problems. As is, it is suitable for XRAY1 problems. However, using `sx=1.25`, `sy=1.5`, and `sz=1.15` makes it suitable for XRAY2 problems.

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

4.7 Biseps

4.8 Utilities

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

4.9 Exceptions

exception tsplib95.exceptions.**TsplibError**
Bases: `Exception`

Base exception for all tsplib95 errors.

exception tsplib95.exceptions.**ParsingError**
Bases: *tsplib95.exceptions.TsplibError*, `ValueError`

Exception raised when a value cannot be parsed from the text.

exception tsplib95.exceptions.**RenderingError**
Bases: *tsplib95.exceptions.TsplibError*, `ValueError`

Exception raised when a value cannot be rendered into text.

exception tsplib95.exceptions.**ValidationError**

Bases: *tsplib95.exceptions.TsplibError*

Exception raised when a problem fails validation.

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 0.7.1 (2020-05-08)

- Bugfix for `StandardProblem.get_nodes` ignoring node indices specified in demands

7.2 0.7.0 (2020-04-18)

- Refactored the models to unify the `Problem` and `Solution` classes into the new `StandardProblem` class.
- 93% test coverage, including distance functions, parsing functions, and rendering functions.
- You can finally *write* problems in TSPLIB95 format! Render to text, write to file, or save to filepath.
- Parsing text, reading files, and loading filepaths are all now supported.
- Deprecated the old loading utils.
- Custom problems now supported by allowing you to define your own fields.
- Library exceptions for parsing and rendering.
- Numerous bugfixes for the distance functions (ATT, XRAY*, GEO).
- Improved the CLI to use a pager and proper column tabulation.
- Made some progress modernizing the FORTRAN code for xray problems.
- Added codecoverage metrics and badge.

7.3 0.6.1 (2020-01-04)

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

7.4 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

7.5 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

7.6 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

7.7 0.3.3 (2019-03-24)

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

7.8 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

7.9 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

7.10 0.3.0 (2018-08-12)

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

7.11 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

7.12 0.1.0 (2018-08-12)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

H

I

J

L

M

N

P

R

S

T

U

V

X