Zhang-Shasha

Release 1.2.0

Mar 12, 2018

Contents

1 Installation **3**

2 Usage **5**

3 API **7**

4 Examples **9**
- 4.1 Tree Format and Usage 9
- 4.2 Specifying Custom Tree Formats 10

5 References **13**

Python Module Index **15**

The `zss` module provides a function (`zss.distance()`) that computes the edit distance between the two given trees, as well as a small set of utilities to make its use convenient.

If you'd like to learn more about how it works, see *References*.

Brought to you by Tim Henderson (tim.tadh@gmail.com).

Get the source or report issues on Github.

CHAPTER 1

Installation

You can get *zss* and its soft requirements (`editdist` and `numpy` >= 1.7) from PyPI:

```
pip install zss
```

Both modules are optional. `editdist` uses string edit distance to compare node labels rather than a simple equal/not-equal check, and `numpy` significantly speeds up the library. The only reason version 1.7 of `numpy` is required is that earlier versions have trouble installing on current versions of Mac OS X.

You can install *zss* from the source code without dependencies in the usual way:

```
python setup.py install
```

If you want to build the docs, you'll need to install Sphinx >= 1.0.

CHAPTER 2

Usage

To compare the distance between two trees, you need:

1. A tree.
2. Another tree.
3. A node-node distance function. By default, *zss* compares the edit distance between the nodes' labels. *zss* currently only knows how to handle nodes with string labels.
4. Functions to let *zss.simple_distance()* traverse your tree.

Here is an example using the library's built-in default node structure and edit distance function:

```
from zss import simple_distance, Node

A = (
    Node("f")
        .addkid(Node("a")
            .addkid(Node("h"))
            .addkid(Node("c")
                .addkid(Node("l"))))
        .addkid(Node("e"))
    )
B = (
    Node("f")
        .addkid(Node("a")
            .addkid(Node("d"))
            .addkid(Node("c")
                .addkid(Node("b"))))
        .addkid(Node("e"))
    )
assert simple_distance(A, B) == 2
```

See more examples

CHAPTER 3

API

zss.**simple_distance**(*A*, *B*, *get_children=zss.Node.get_distance*, *get_label=zss.Node.get_label*, *label_dist=strdist*)

Computes the exact tree edit distance between trees A and B.

Use this function if both of these things are true:

- The cost to insert a node is equivalent to `label_dist('', new_label)`
- The cost to remove a node is equivalent to `label_dist(new_label, '')`

Otherwise, use *zss.distance()* instead.

Parameters

- **A** – The root of a tree.
- **B** – The root of a tree.
- **get_children** – A function `get_children(node) == [node children]`. Defaults to *zss.Node.get_children()*.
- **get_label** – A function `get_label(node) == 'node label'`.All labels are assumed to be strings at this time. Defaults to *zss.Node.get_label()*.
- **label_dist** – A function `label_distance((get_label(node1), get_label(node2)) >= 0`. This function should take the output of `get_label(node)` and return an integer greater or equal to 0 representing how many edits to transform the label of `node1` into the label of `node2`. By default, this is string edit distance (if available). 0 indicates that the labels are the same. A number N represent it takes N changes to transform one label into the other.
- **return_operations** – if True, return a tuple (cost, operations) where operations is a list of the operations to transform A into B.

Returns An integer distance [0, inf+)

zss.**distance**(*A*, *B*, *get_children*, *insert_cost*, *remove_cost*, *update_cost*, *return_operations=False*)

Computes the exact tree edit distance between trees A and B with a richer API than *zss.simple_distance()*.

Use this function if either of these things are true:

- The cost to insert a node is **not** equivalent to the cost of changing an empty node to have the new node's label
- The cost to remove a node is **not** equivalent to the cost of changing it to a node with an empty label

Otherwise, use *zss.simple_distance()*.

Parameters

- **A** – The root of a tree.
- **B** – The root of a tree.
- **get_children** – A function `get_children(node) == [node children]`. Defaults to *zss.Node.get_children()*.
- **insert_cost** – A function `insert_cost(node) == cost to insert node >= 0`.
- **remove_cost** – A function `remove_cost(node) == cost to remove node >= 0`.
- **update_cost** – A function `update_cost(a, b) == cost to change a into b >= 0`.
- **return_operations** – if True, return a tuple (cost, operations) where operations is a list of the operations to transform A into B.

Returns An integer distance [0, inf+)

class `zss.`**Node**(*label*, *children=None*)

A simple node object that can be used to construct trees to be used with *zss.distance()*.

Example:

```
Node("f")
    .addkid(Node("a")
        .addkid(Node("h"))
        .addkid(Node("c")
            .addkid(Node("l"))))
    .addkid(Node("e"))
```

addkid(*node*, *before=False*)

Add the given node as a child of this node.

get(*label*)

Returns Child with the given label.

static get_children(*node*)

Default value of `get_children` argument of *zss.distance()*.

Returns `self.children`.

static get_label(*node*)

Default value of `get_label` argument of *zss.distance()*.

Returns `self.label`.

iter()

Iterate over this node and its children in a preorder traversal.

CHAPTER 4

Examples

4.1 Tree Format and Usage

By default, the tree is represented by objects referencing each other. Each node is represented by an object with the attributes `label` and `children`, where `label` is a string and `children` is a list of other objects. However, all of this is configurable by passing in functions. Here is how to use the default API:

To find the distance between two object trees, call `zss.simple_distance(root1, root2)`.

The object format is used by the tests and is probably the easiest to work with.

4.1.1 A simple example

```
from zss import simple_distance, Node

# Node(label, children)
# a---> b
#  \--> c
c = Node('c', [])
b = Node('b', [])
a = Node('a', [b, c])
assert simple_distance(a, a) == 0

# a---> c
a2 = Node('a', [Node('c', [])])
assert simple_distance(a, a2) == 1
```

4.1.2 Another Example:

```
from zss import simple_distance, Node

A = (
```

```
    Node("f")
        .addkid(Node("a")
            .addkid(Node("h"))
            .addkid(Node("c")
                .addkid(Node("l"))))
        .addkid(Node("e"))
    )
B = (
    Node("f")
        .addkid(Node("a")
            .addkid(Node("d"))
            .addkid(Node("c")
                .addkid(Node("b"))))
        .addkid(Node("e"))
    )
assert simple_distance(A, B) == 2
```

See `test_metricspace.py` for more examples.

4.2 Specifying Custom Tree Formats

Specifying custom tree formats and distance metrics is easy. The *`zss.simple_distance()`* function takes 3 extra parameters besides the two tree to compare:

1. `get_children` - a function to retrieve a list of children from a node.
2. `get_label` - a function to retrieve the label object from a node.
3. `label_dist` - a function to compute the non-negative integer distance between two node labels.

4.2.1 Example

```
#!/usr/bin/env python

import zss

try:
    from editdist import distance as strdist
except ImportError:
    def strdist(a, b):
        if a == b:
            return 0
        else:
            return 1

def weird_dist(A, B):
    return 10*strdist(A, B)

class WeirdNode(object):

    def __init__(self, label):
        self.my_label = label
        self.my_children = list()

    @staticmethod
```

```
    def get_children(node):
        return node.my_children

    @staticmethod
    def get_label(node):
        return node.my_label

    def addkid(self, node, before=False):
        if before:  self.my_children.insert(0, node)
        else:   self.my_children.append(node)
        return self

A = (
WeirdNode("f")
    .addkid(WeirdNode("d")
    .addkid(WeirdNode("a"))
    .addkid(WeirdNode("c")
        .addkid(WeirdNode("b"))
    )
    )
    .addkid(WeirdNode("e"))
)
B = (
WeirdNode("f")
    .addkid(WeirdNode("c")
    .addkid(WeirdNode("d")
        .addkid(WeirdNode("a"))
        .addkid(WeirdNode("b"))
    )
    )
    .addkid(WeirdNode("e"))
)

dist = zss.simple_distance(
    A, B, WeirdNode.get_children, WeirdNode.get_label, weird_dist)

print dist
assert dist == 20
```

CHAPTER 5

References

The algorithm used by `zss` is taken directly from the original paper by Zhang and Shasha. If you would like to discuss the paper, or the the tree edit distance problem (we have implemented a few other algorithms as well) please email the authors.

approxlib by Dr. Nikolaus Augstent contains a good Java implementation of Zhang-Shasha as well as a number of other useful tree distance algorithms.

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing, 18:1245–1262, 1989. (the original paper)

Slide deck overview of Zhang-Shasha

Another paper describing Zhang-Shasha

Python Module Index

z

zss, 7

Index

A

addkid() (zss.Node method), 8

D

distance() (in module zss), 7

G

get() (zss.Node method), 8
get_children() (zss.Node static method), 8
get_label() (zss.Node static method), 8

I

iter() (zss.Node method), 8

N

Node (class in zss), 8

S

simple_distance() (in module zss), 7

Z

zss (module), 7