CoDist

Release 0.2.0

James Finnie-Ansley

Apr 08, 2024

CONTENTS:

CHAPTER

ONE

API

codist	Base codist module.
codist.ast	Utilities for converting abstract syntax trees to codist.tree.Tree objects
codist.distance	Functions to compute the edit distance between trees with given cost functions.
codist.tree	Tree utilities and type definitions.

1.1 codist

Base codist module. Imports functions for constructing trees and computing edit distances.

class Cost(*delete: ~collections.abc.Callable[[T], float] = <function Cost.<lambda>>, insert: ~collections.abc.Callable[[T], float] = <function Cost.<lambda>>, relabel: ~collections.abc.Callable[[T, T], float] = <function Cost.<lambda>>*)

A set of tree edit cost functions for deleting, inserting and relabelling nodes.

By default, returns 1 except for the case (a -> a) which returns 0

Parameters

- **delete** – A cost function, `(T) -> float` for the change operation `(T -> )`. Default is `(T) -> 1`
- **insert** – A cost function `(T) -> float` for the change operation `( -> T)`. Default is `(T) -> 1`
- **relabel** – A cost function `(T1, T2) -> float` for the change operation `(T1 -> T2)`. Default is `(T1, T2) -> 0 if T1 == T2 else 1`

Variables

- **delete** – A cost function, `(T) -> float`
- **insert** – A cost function `(T) -> float`
- **relabel** – A cost function `(T1, T2) -> float`

t(*root: T*, **children: Tree*) → Tree

Small convenience function to help construct trees

tree_dist(*tree1: Tree*, *tree2: Tree*, *cost: ~codist.distance.Cost = <codist.distance.Cost object>*) → float

Tree edit cost using the given cost function.

Parameters

- **tree1** – the initial tree
- **tree2** – the target tree
- **cost** – a Cost object defining cost functions

Returns
The edit distance between `tree1` and `tree2`

tree_edit(*tree1: Tree*, *tree2: Tree*, *cost: ~codist.distance.Cost = <codist.distance.Cost object>*) → tuple[float, tuple[Change, ...]]

Tree edit cost and edit path using the given cost function.

Parameters

- **tree1** – the initial tree
- **tree2** – the target tree
- **cost** – a Cost object defining cost functions

Returns
A tuple containing the edit distance between `tree1` and `tree2` and a tuple of *Change* operations where each change operation is a 3-tuple of the form `(T | Lambda -> T | Lambda, ctx)` where `Lambda` is a singleton string: `""`

1.2 codist.ast

Utilities for converting abstract syntax trees to *codist.tree.Tree* objects

Functions

ast_silhouette(node)	Returns a Tree containing only AST node type information
parse_ast_silhouette(code)	Parses the given code and returns a Tree containing only AST node type information.

ast_silhouette(*node: AST*) → Tree[str]

Returns a Tree containing only AST node type information

parse_ast_silhouette(*code: str*) → Tree[str]

Parses the given code and returns a Tree containing only AST node type information.

1.3 codist.distance

Functions to compute the edit distance between trees with given cost functions.

Module Attributes

Change	A change operation of the form `(T | Lambda -> T | Lambda, ctx)` Where `ctx` is either an index or Lambda providing some context for the change operation.

Functions

tree_dist(tree1, tree2[, cost])	Tree edit cost using the given cost function.
tree_edit(tree1, tree2[, cost])	Tree edit cost and edit path using the given cost function.

Classes

Cost(delete, float] = >, insert, float] = >, ...)	A set of tree edit cost functions for deleting, inserting and relabelling nodes.

Change: type[tuple[T | , T | , int | Lambda]] = Change

A change operation of the form `(T | Lambda -> T | Lambda, ctx)` Where `ctx` is either an index or Lambda providing some context for the change operation.

- For insertions (-> T, ctx), the ctx is the postorder index of the parent node in tree2 that T is being added under
- For deletions (T -> , ctx), the ctx is the postorder index of the node in tree1 that is deleted
- For relabelings (T1 -> T2, ctx), the ctx is the postorder index of the node in tree1 that is relabeled

---

Note: The ctx variable provides some context for change operations, but does not provide, for example, the indices of the siblings that are inserted as children of T for an insertion operation.

---

class Cost(*delete: ~collections.abc.Callable[[T], float] = <function Cost.<lambda>>, insert: ~collections.abc.Callable[[T], float] = <function Cost.<lambda>>, relabel: ~collections.abc.Callable[[T, T], float] = <function Cost.<lambda>>*)

A set of tree edit cost functions for deleting, inserting and relabelling nodes.

By default, returns 1 except for the case (a -> a) which returns 0

Parameters

- **delete** – A cost function, `(T) -> float` for the change operation `(T -> )`. Default is `(T) -> 1`
- **insert** – A cost function `(T) -> float` for the change operation `( -> T)`. Default is `(T) -> 1`
- **relabel** – A cost function `(T1, T2) -> float` for the change operation `(T1 -> T2)`. Default is `(T1, T2) -> 0 if T1 == T2 else 1`

Variables

- **delete** – A cost function, `(T) -> float`

- **insert** – A cost function `(T) -> float`
- **relabel** – A cost function `(T1, T2) -> float`

tree_dist(*tree1: Tree*, *tree2: Tree*, *cost: ~codist.distance.Cost = <codist.distance.Cost object>*) → float

Tree edit cost using the given cost function.

Parameters

- **tree1** – the initial tree
- **tree2** – the target tree
- **cost** – a Cost object defining cost functions

Returns

The edit distance between `tree1` and `tree2`

tree_edit(*tree1: Tree*, *tree2: Tree*, *cost: ~codist.distance.Cost = <codist.distance.Cost object>*) → tuple[float, tuple[Change, ...]]

Tree edit cost and edit path using the given cost function.

Parameters

- **tree1** – the initial tree
- **tree2** – the target tree
- **cost** – a Cost object defining cost functions

Returns

A tuple containing the edit distance between `tree1` and `tree2` and a tuple of *Change* operations where each change operation is a 3-tuple of the form `(T | Lambda -> T | Lambda, ctx)` where `Lambda` is a singleton string: `""`

1.4 codist.tree

Tree utilities and type definitions.

Module Attributes

Lambda	A singleton used in change operations
Tree	A tree type.

Functions

keyroots(tree)	Postorder traversal of keyroot indices for keyroots in T
leftmosts(tree)	The postorder traversal of $l(i)$ for each index i in T
parents(tree)	The postorder enumeration of the indices of the parent of each node, The root of the tree has the parent Lambda (i.e. indicating no parent).
postorder(tree)	A postorder traversal of the node data in `tree`
t(root, *children)	Small convenience function to help construct trees

Lambda: Final[str] = ''

A singleton used in change operations

Tree: type[tuple[T, tuple[Tree[T], ...]]] = Tree

A tree type. A tree is any tuple of the form: `Tree[T] = tuple[T, tuple[Tree[T], ...]]`

keyroots(*tree: Tree*) → tuple[int, ...]

Postorder traversal of keyroot indices for keyroots in T

leftmosts(*tree: Tree*) → tuple[int, ...]

The postorder traversal of $l(i)$ for each index i in T

parents(*tree: Tree*) → tuple[int | Lambda, ...]

The postorder enumeration of the indices of the parent of each node, The root of the tree has the parent Lambda (i.e. indicating no parent)

postorder(*tree: Tree*) → tuple[T, ...]

A postorder traversal of the node data in `tree`

t(*root: T*, **children: Tree*) → Tree

Small convenience function to help construct trees

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX