majorminer

Dominic Plein, Julien Meier

Feb 13, 2022

CONTENTS

Welcome to the documentation of the majorminer research project realized by two students of the DHBW Karlsruhe. This documentation only includes the API for the Python code, NOT the C++ code.

- genindex
- modindex

CHAPTER

ONE

OVERVIEW

1.1 Goal

1.2 Python architecture

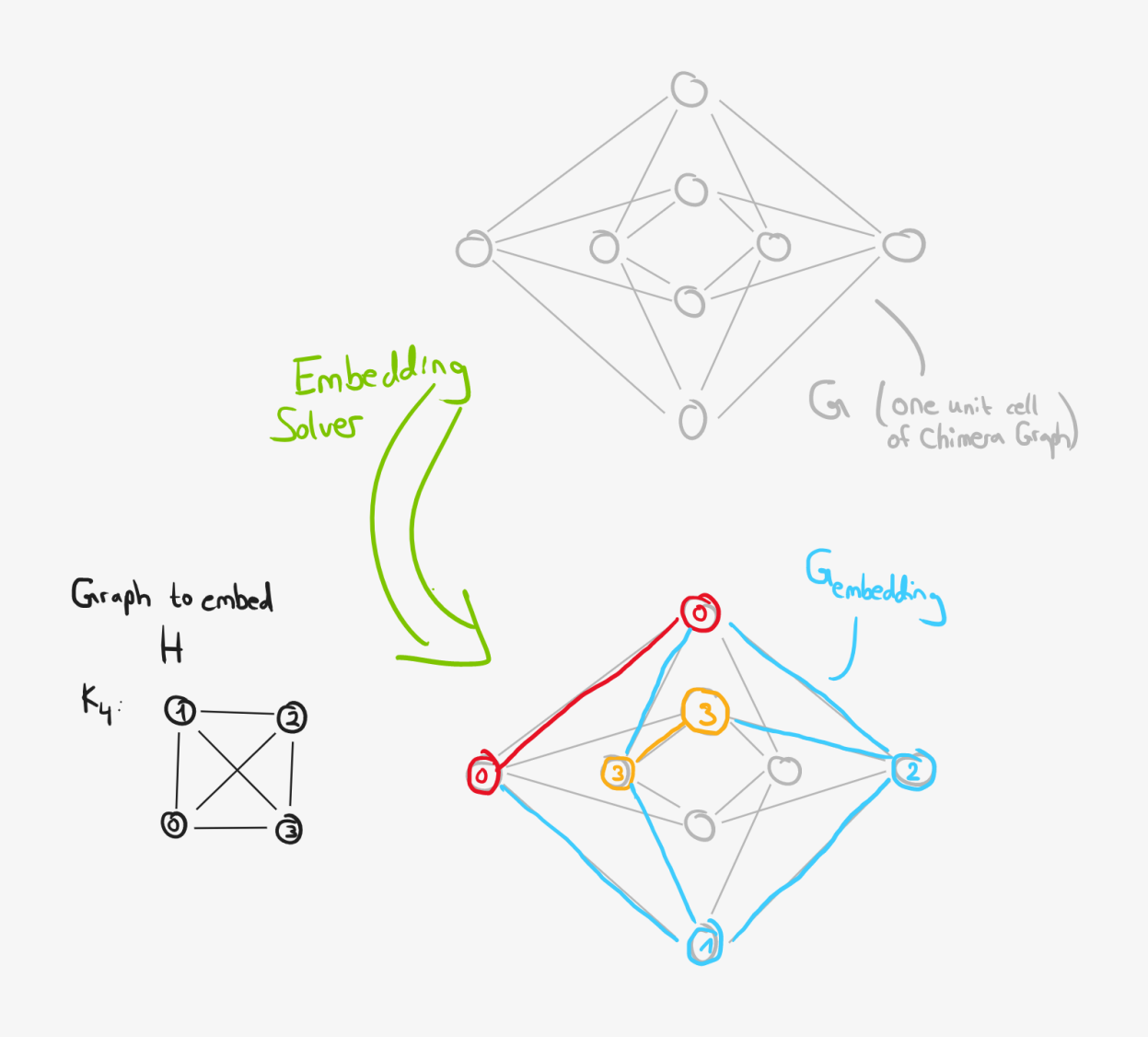

Fig. 1: Principle of the embedding solver. Find an embedding for minor H in graph G (in this case: a simple Chimera Graph). Colors indicate different chains (and are not related to the colors in the architecture diagram below).

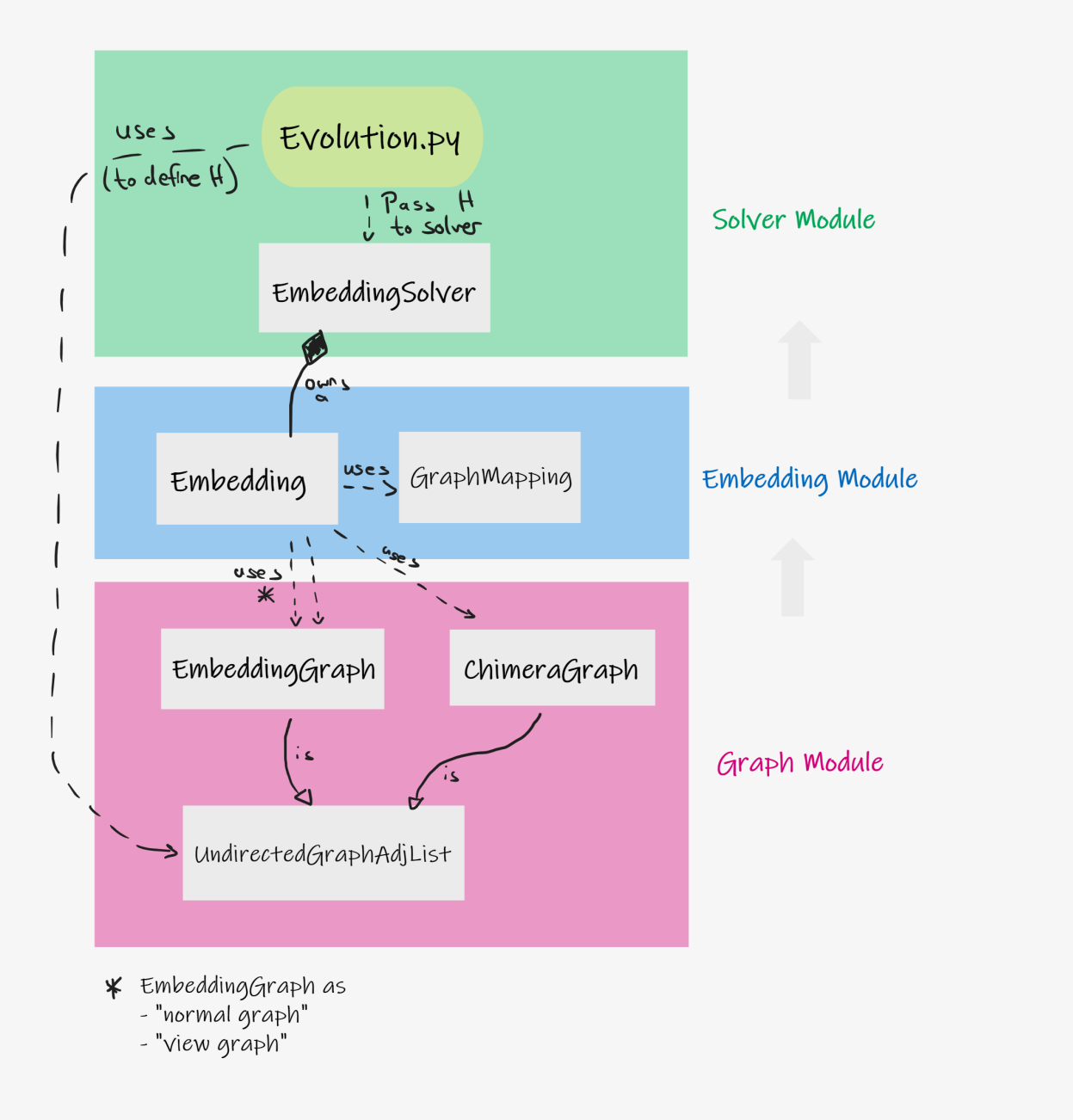

Fig. 2: Simplified overview over the Python architecture. Grey boxes are classes.

CHAPTER

TWO

API REFERENCE

2.1 API Reference

2.1.1 Graph Module

class src.graph.undirected_graph.**AdjListEntryWithCosts**

Bases: object

One entry of an adjacency list.

Describes edges to other nodes and assigns costs to these edges.

> **Warning:** This AdjacencyList does *not* check for invalid indexing. Expect KeyErrors to arise if the caller does not handle indexing correctly.

Note: It is guaranteed that an edge from node u to node v cannot exist multiple times with different costs.

---

get_neighbors() → list[int]

Returns all neighboring nodes.

Returns All neighboring nodes (from the current node).

Return type list[int]

get_neighbors_with_costs() → list[tuple[int, int]]

Returns all neighboring nodes with costs (from the current node).

Returns All neighboring nodes (from the current node) with costs.

Return type list[tuple[int, int]]

remove_edge_to(*to: int*) → None

Removes an edge to a node.

Parameters **to** (*int*) – Other node to specify the edge that should be removed.

set_edge_to(*to: int*, *cost: int*) → None

Sets an undirected edge with cost to a node.

> **Warning:** This may override an existing edge with new costs.

Parameters

- **to** (*int*) – Other node to specify the edge.
- **cost** (*int*) – Cost for the edge

exception src.graph.undirected_graph.**GraphNodeIndexError**(*nodes_count: int*, *index: int*)
Bases: Exception

Error that occurs when accessing a node that does not exist in the Graph.

Parameters

- **nodes_count** (*int*) – Number of nodes in the Graph.
- **index** (*int*) – Invalid node (represented as integer index in the Graph).

class src.graph.undirected_graph.**UndirectedGraphAdjList**(*nodes_count: int*)
Bases: object

Undirected Graph using an adjacency list.

add_node() → int
Adds a new node to this Graph.

Returns The new node number.

Return type int

exists_edge(*frm: int*, *to: int*) → bool
Checks if an edge between two nodes exists.

Parameters

- **frm** (*int*) – One node of the edge.
- **to** (*int*) – The other node of the edge.

Returns True if the edge exists.

Return type bool

Raises **GraphNodeIndexError** – If the given node frm or to does not exist in the Graph.

get_edges() → list[tuple[int, int, int]]
Returns all edges of this Graph.

Returns A List of entries (frm, to, cost) describing all edges of this Graph with their respective costs.

Return type list[tuple[int, int, int]]

get_neighbor_nodes(*frm: int*) → list[int]
Returns all neighboring nodes.

Neighboring nodes are nodes that are connected via an edge to this node.

Parameters **frm** (*int*) – Node from which the neighboring nodes should be retrieved.

Returns List of integers describing the neighboring nodes.

Return type list[int]

Raises **GraphNodeIndexError** – If the given node frm does not exist in the Graph.

get_neighbor_nodes_with_costs(*frm: int*) → list[tuple[int, int]]
Returns all neighboring nodes with costs.

Neighboring nodes are nodes that are connected via an edge to this node.

Parameters **frm** (*int*) – Node from which the neighboring nodes should be retrieved.

Returns List of tuples (to, cost) describing edges with costs.

Return type list[tuple[int, int]]

Raises **GraphNodeIndexError** – If the given node `frm` does not exist in the Graph.

get_nodes() → list[int]
Returns all nodes of this Graph.

Returns All nodes of this Graph.

Return type list[int]

has_neighbor_nodes(*frm: int*) → bool
Checks if the given node has neighboring nodes.

Neighboring nodes are nodes that are connected via an edge to this node.

Parameters **frm** (*int*) – Node from which the neighboring nodes should be retrieved.

Returns True if the given node has neighboring nodes.

Return type bool

Raises **GraphNodeIndexError** – If the given node `frm` does not exist in the Graph.

remove_all_edges_from_node(*frm: int*) → None
Removes all edges connected to the given node.

Parameters **frm** (*int*) – node from which all edges should be removed.

Raises **GraphNodeIndexError** – If the given node `frm` does not exist in the Graph.

remove_edge(*frm: int*, *to: int*) → None
Removes the edge (if present) between nodes `frm` and `to`.

Parameters

- **frm** (*int*) – One node of the edge.
- **to** (*int*) – The other node of the edge.

Raises **GraphNodeIndexError** – If the given node `frm` or `to` does not exist in the Graph.

set_edge(*frm: int*, *to: int*, *cost=0*) → None
Sets an edge between nodes `frm` and `to` and assigns the given `cost`.

> **Warning:** This may override an existing edge with new costs.

Parameters

- **frm** (*int*) – One node of the edge.
- **to** (*int*) – The other node of the edge.
- **cost** (*int*) – The costs for the edge

Raises **GraphNodeIndexError** – If the given node `frm` or `to` does not exist in the Graph.

class src.graph.embedding_graph.**EmbeddingGraph**(*nodes_count*)
Bases: src.graph.undirected_graph.UndirectedGraphAdjList

A Graph to **form** an embedding (using an adjacency list).

This Graph is supposed to **converge into a valid minor** of another graph by using the methods it provides, e.g. to embed edges, form chains etc.

Hint: Pay attention whether you use methods which are directly defined here or in the parent class src.graph.undirected_graph.UndirectedGraphAdjList. Methods defined over there may not be convenient as the current class tries to hide implementation details, e.g. chains being encoded as edge costs. It is recommend to just stick to the methods defined here (in the subclass).

Warning: This class does not implement any checks for chains, so it is possible to assign a node to multiple chains.

add_chain(*node1: int*, *node2: int*) → int
Adds a new chain between two nodes.

Parameters

- **node1** (*int*) – The first node of the chain.
- **node2** (*int*) – The second node of the chain.

Returns The new chain's identifier.

Return type int

embed_edge(*frm: int*, *to: int*, *chain=0*) → None
Embeds an edge (and optionally assigns a chain).

Parameters

- **frm** (*int*) – One node of the edge.
- **to** (*int*) – The other node of the edge.
- **chain** (*int, optional*) – The chain to assign to the new embedded edge. The default value *0* means: no chain, just a "normal" edge.

Raises **GraphNodeIndexError** – If the given node frm or to does not exist in the Graph.

get_chain_nodes(*chain: int*) → list[int]
Returns all nodes that are contained in a given chain.

Parameters **chain** (*int*) – The chain to retrieve all nodes for.

Returns All nodes that are contained in *chain*. (These nodes might be contained in other chains at the same time.)

Return type list[int]

get_embedded_edges() → list[tuple[int, int, int]]
Returns all embedded edges.

Returns A List of entries (frm, to, cost) describing all embedded edges of this Graph with their respective costs.

Return type list[tuple[int, int, int]]

get_embedded_nodes() → list[int]
Returns all embedded nodes.

Returns The embedded nodes.

Return type list[int]

get_embedding() → tuple[list[int], list[tuple[int, int, int]]]
Returns the current embedding.

Returns A tuple of nodes and edges. *nodes* is a list of integers, while *edges* is a list of tuples (frm, to, chain) describing the edges (frm node, to node, chain)

Return type tuple[list[int], list[tuple[int, int, int]]]

get_node_chains(*node: int*) → list[int]
Returns all chains in which the current node is contained.

A node might be contained in multiple chains. This would be an incorrect embedding in any case but allows for evolutionary algorithms to have temporarily invalid states which may be beneficial for some algorithms.

Parameters **node** (*int*) – The node for which the chains are to be determined.

Returns All chains in which *node* is contained.

Return type list[int]

get_nodes_in_same_chains(*node: int*) → list[int]
Returns the nodes that are in one of the chains the given node is contained in.

> **Warning:** A node might be contained in multiple chains temporarily. Therefore, this method might return nodes that are not related to each other as they are located in different chains.
>
> Consider the functions *get_node_chains()* and *get_chain_nodes()* as alternatives.

Parameters **node** (*int*) – The node from which to take the chain to compare against.

Returns All nodes that are in one of the chains *node* is contained in.

Return type list[int]

class src.graph.chimera_graph.**ChimeraGraphLayout**
Bases: src.graph.undirected_graph.UndirectedGraphAdjList

A Chimera Graph representation.

Note: As of now, this Graph is limited to one single unit cell with shore size 4.

2.1.2 More Modules

Note: To come soon…

PYTHON MODULE INDEX

S

INDEX

U