Activity Browser Documentation

Release 0.1.1

Bernhard Steubing

September 02, 2015

Contents

1 Features **3**

2 Extensions **5**

3 Documentation **7**
3.1 Installation / Source Code 7
3.2 Modular LCA approach 7

4 Class reference **11**
4.1 Class reference 11

5 License **17**

6 Contact **19**

7 Indices and tables **21**

The Activity Browser is a free LCA software. It builds upon brightway2 for much of its functionality (e.g. LCA calculations). It extends brightway2 through a graphical user interface (GUI) increasing the efficiency of certain tasks.

As it is open source, you can add your own extensions.

CHAPTER 1

Features

Core features currently involve:

- a graphical user interface (GUI) to brightway2
- fast browser-inspired navigation through inventory databases
- creating and modifying inventories and databases
- fast LCA calculations (even multi-inventory-multi-method)

CHAPTER 2

Extensions

Modular LCA approach - a tool for modeling life cycles based on *modules*.

CHAPTER 3

Documentation

3.1 Installation / Source Code

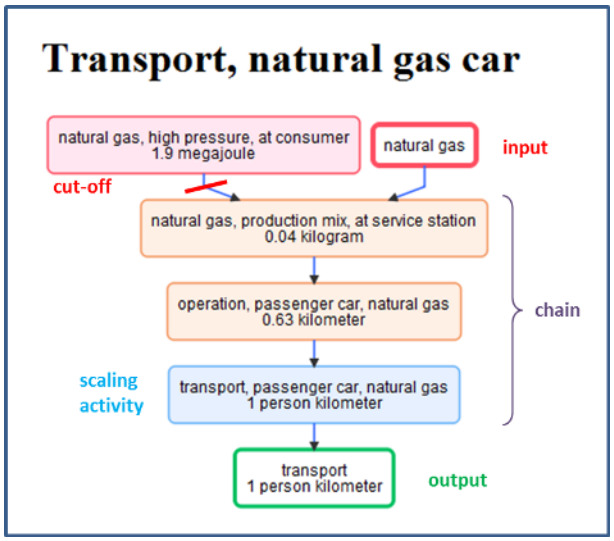
The python source code is available at bitbucket.

3.2 Modular LCA approach

Using Modules (formerly called 'Meta-Processes') to model life cycle inventories enables:

- modeling life cycle stages based on unit processes from an LCI database
- linking modules to represent complete, possibly new life cycles
- efficient modeling of alternative life cycles
- efficient coupling of LCA and optimization

3.2.1 Modules



Modules can group several life cycle inventories into a single process.

Modules *need* to have:

- a name
- at least one product output with a user defined name
- at least one activity
- a scaling activity (determined autmatically)

Modules *can* have:

- additional processes forming the process chain
- multiple outputs
- multiple inputs; product inputs involve a cut-off

3.2.2 Data Format of Modules

Modules can be specified in the format shown below. It is used to define and store modules. All other properties are calculated based on this data, e.g. scaling of edges, LCA results, etc. using the methods of the MetaProcess class.

```
example_module = {
        'name': 'user defined module name',
        'outputs': [
            (key, 'user defined product name', user defined amount),
        ],
        'chain': [
            (key),
        ],
        'cuts': [
            (parent_key, child_key, 'user defined input product name', amount),
        ],
        'output_based_scaling': True,
}
```

Notes:

***Keys*:**

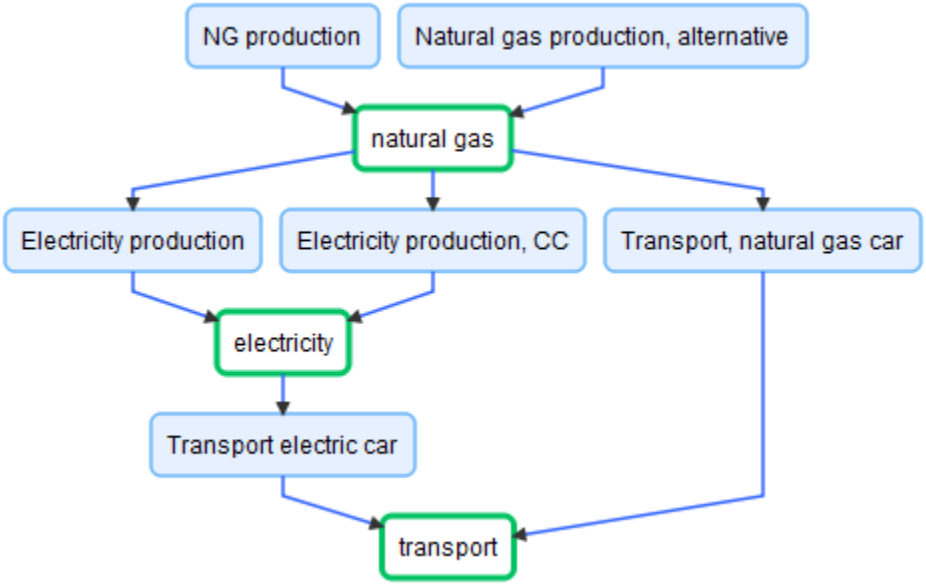
- Keys are a tuple composed of two elements, where the first refers to the database and the second to the activity, thus ('database name', 'module name or uuid')

***Output-based scaling*:**

- The default value is *True*. If set to *False*, the scaling activities will be scaled to 1.0 no matter how the product outputs are defined by the user. This can be used to

1. to create artificial outputs that are not part of the original dataset (the user needs to see whether that makes sense)
2. when ecoinvent 2.2 multi-output activities, as imported in brightway2, are used, as these don't include the output products, which need to be manually defined.

3.2.3 Linked Modules



Linked Module Systems are created by combining modules based on their product inputs and outputs. As shown in the example, the product based linking allows to efficiently specify alternative supply chains.

A detailed description of the math behind modules, its application scope and examples are provided in the following paper (reference will be available soon).

CHAPTER 4

Class reference

4.1 Class reference

4.1.1 Modules (formerly called 'meta-processes')

class `metaprocess.`**`MetaProcess`**(*name*, *outputs*, *chain*, *cuts*, *output_based_scaling=True*, ***kwargs*)

A description of one or several processes from a life cycle inventory database. It has the following characteristics:

- It produces one or several output products
- It has at least one process from an inventory database
- It has one or several scaling activities that are linked to the output of the system. They are calculated automatically based on the product output (exception: if output_based_scaling=False, see below).
- Inputs may be cut-off. Cut-offs are remembered and can be used in a linked meta-process to recombine meta-processes to form supply chains (or several, alternative supply chains).

Args:

- *name* (`str`): Name of the meta-process
- *outputs* (`[(key, str, optional float)]`): A list of products produced by the meta-process. Format is `(key into inventory database, product name, optional amount of product produced)`.
- *chain* (`[key]`): A list of inventory processes in the supply chain (not necessarily in order).
- *cuts* (`[(parent_key, child_key, str, float)]`): A set of linkages in the supply chain that should be cut. These will appear as **negative** products (i.e. inputs) in the process-product table. The float amount is determined automatically. Format is (input key, output key, product name, amount).
- *output_based_scaling* (`bool`): True: scaling activities are scaled by the user defined product outputs. False: the scaling activities are set to 1.0 and the user can define any output. This may not reflect reality or original purpose of the inventory processes.

`construct_graph`(*db*)

Construct a list of edges.

Args:

- *db* (dict): The supply chain database

Returns: A list of (in, out, amount) edges.

getFilteredDatabase(*depending_databases*, *chain*)
Extract the supply chain for this process from larger database.

Args:

- *nodes* (set): The datasets to extract (keys in db dict)
- *db* (dict): The inventory database, e.g. ecoinvent

Returns: A filtered database, in the same dict format

getScalingActivities(*chain*, *edges*)
Which are the scaling activities (at least one)?

Calculate by filtering for processes which are not used as inputs.

Args:

- *chain* (set): The supply chain processes
- *edges* (list): The list of supply chain edges

Returns: Boolean isSimple, List heads.

get_edge_lists()
Get lists of external and internal edges with original flow values or scaled to the meta-process.

get_product_inputs_and_outputs()
Returns a list of product inputs and outputs.

get_supply_vector(*chain*, *edges*, *scaling_activities*, *outputs*)
Construct supply vector (solve linear system) for the supply chain of this simplified product system.

Args:

- *chain* (list): Nodes in supply chain
- *edges* (list): List of edges
- *scaling_activities* (key): Scaling activities

Returns: Mapping from process keys to supply vector indices Supply vector (as list)

lca(*method*, *amount=1.0*, *factorize=False*)
Calculates LCA results for a given LCIA method and amount. Returns the LCA score.

mp_data
Returns a dictionary of meta-process data as specified in the data format.

pad_cuts()
Makes sure that each cut includes the amount that is cut. This is retrieved from self.internal_scaled_edges_with_cuts.

pad_outputs(*outputs*)
If not given, adds default values to outputs:

•output name: “Unspecified Output”

•amount: 1.0

Args:

- *outputs* (list): outputs

Returns: Padded outputs

pp
Property shortcut for returning a list of product intputs and outputs.

remove_cuts_from_chain(*chain*, *cuts*)
Remove chain items if they are the parent of a cut. Otherwise this leads to unintended LCIA results.

save_as_bw2_dataset(*db_name='MP default'*, *unit=None*, *location=None*, *categories=[]*, *save_aggregated_inventory=False*)
Save simplified process to a database.

Creates database if necessary; otherwise *adds* to existing database. Uses the `unit` and `location` of `self.scaling_activities[0]`, if not otherwise provided. Assumes that one unit of the scaling activity is being produced.

Args:

- *db_name* (str): Name of Database
- *unit* (str, optional): Unit of the simplified process
- *location* (str, optional): Location of the simplified process
- *categories* (list, optional): Category/ies of the scaling activity
- *save_aggregated_inventory* (bool, optional): Saves in output minus input style by default (True), otherwise aggregated inventory of all inventories linked within the meta-process

4.1.2 Linked Modules (formerly called 'linked Meta-Processes')

class `linkedmetaprocess.`**LinkedMetaProcessSystem**(*mp_list=None*)
A linked meta-process system holds several interlinked meta-processes. It has methods for:

•loading / saving linked meta-process systems

•returning information, e.g. product and process names, the product-process matrix

•determining all alternatives to produce a given functional unit

•calculating LCA results for individual meta-processes

•calculating LCA results for a demand from the linked meta-process system (possibly for all alternatives)

Meta-processes *cannot* contain:

•2 processes with the same name

•identical names for products and processes (recommendation is to capitalize process names)

Args:

•*mp_list* (`[MetaProcess]`): A list of meta-processes

add_mp(*mp_list*, *rename=False*)
Adds meta-processes to the linked meta-process system.

mp_list can contain meta-process objects or the original data format used to initialize meta-processes.

all_pathways(*functional_unit*)
Returns all alternative pathways to produce a given functional unit. Data output is a list of lists. Each sublist contains one path made up of products and processes. The input Graph may not contain cycles. It may contain multi-output processes.

Args:

•*functional_unit*: output product

edges(*mp_list=None*)
Returns an edge list for all edges within the linked meta-process system.

mp_list can be a list of meta-process objects or meta-process names.

get_cut_names(*mp_list=None*)
Returns cut/input product names for a list of meta-processes.

get_output_names(*mp_list=None*)
Returns output product names for a list of meta-processes.

get_pp_matrix(*mp_list=None*)
Returns the product-process matrix as well as two dictionaries that hold row/col values for each product/process.

mp_list can be used to limit the scope to the contained processes

get_process_names(*mp_list=None*)
Returns a the names of a list of meta-processes.

get_processes(*mp_list=None*)
Returns a list of meta-processes.

mp_list can be a list of meta-process objects or meta-process names.

get_product_names(*mp_list=None*)
Returns the output and input product names of a list of meta-processes.

mp_list can be a list of meta-process objects or meta-process names.

lca_alternatives(*method*, *demand*)
Calculation of LCA results for all alternatives in a linked meta-process system that yield a certain demand. Results are stored in a list of dictionaries as described in 'lca_linked_processes'.

Args:

•*method*: LCIA method

•*demand* (dict): keys: product names, values: amount

lca_linked_processes(*method*, *process_names*, *demand*)
Performs LCA for a given demand from a linked meta-process system. Works only for square matrices (see scaling_vector_foreground_demand).

Returns a dictionary with the following keys:

•*path*: involved process names

•*demand*: product demand

•*scaling vector*: result of the demand

•*LCIA method*: method used

•*process contribution*: contribution of each process

•*relative process contribution*: relative contribution

•*LCIA score*: LCA result

Args:

•*method*: LCIA method

•*process_names*: selection of processes from the linked meta-process system (that yields a square matrix)

•*demand* (dict): keys: product names, values: amount

lca_processes(*method*, *process_names=None*, *factorize=False*)
Returns a dictionary where *keys* = meta-process name, *value* = LCA score

load_from_file(*filepath*, *append=False*)
Loads a meta-process database, makes a MetaProcess object from each meta-process and adds them to the linked meta-process system.

Args:

•filepath: file path

•append: adds loaded meta-processes to the existing database if True

processes
Returns all process names.

product_process_dict(*mp_list=None*, *process_names=None*, *product_names=None*)
Returns a dictionary that maps meta-processes to produced products (key: product, value: meta-process). Optional arguments `mp_list`, `process_names`, `product_names` can used as filters.

products
Returns all product names.

remove_mp(*mp_list*)
Remove meta-processes from the linked meta-process system.

mp_list can be a list of meta-process objects or meta-process names.

save_to_file(*filepath*)
Saves data for each meta-process in the meta-process data format using pickle and updates the linked meta process system.

scaling_vector_foreground_demand(*mp_list*, *demand*)
Returns a scaling dictionary for a given demand and matrix defined by a list of processes (or names). Keys: process names. Values: scaling vector values.

Args:

•*mp_list*: meta-process objects or names

•*demand* (dict): keys: product names, values: amount

update(*mp_list*)
Updates the linked meta-process system every time processes are added, modified, or deleted. Errors are thrown in case of:

•identical names for products and processes

•identical names of different meta-processes

•if the input is not of type MetaProcess()

update_name_map()
Updates the name map, which maps product names (outputs or cuts) to activity keys. This is used in the Activity Browser to automatically assign a product name to already known activity keys.

upstream_products_processes(*product*)
Returns all upstream products and processes related to a certain product (functional unit).

CHAPTER 5

License

The project is licensed under the GNU General Public License.

CHAPTER 6

Contact

Bernhard Steubing

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

A

add_mp() (linkedmetaprocess.LinkedMetaProcessSystem method), 13
all_pathways() (linkedmetaprocess.LinkedMetaProcessSystem method), 13

C

construct_graph() (metaprocess.MetaProcess method), 11

E

edges() (linkedmetaprocess.LinkedMetaProcessSystem method), 13

G

get_cut_names() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
get_edge_lists() (metaprocess.MetaProcess method), 12
get_output_names() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
get_pp_matrix() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
get_process_names() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
get_processes() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
get_product_inputs_and_outputs() (metaprocess.MetaProcess method), 12
get_product_names() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
get_supply_vector() (metaprocess.MetaProcess method), 12
getFilteredDatabase() (metaprocess.MetaProcess method), 12
getScalingActivities() (metaprocess.MetaProcess method), 12

L

lca() (metaprocess.MetaProcess method), 12
lca_alternatives() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
lca_linked_processes() (linkedmetaprocess.LinkedMetaProcessSystem method), 14
lca_processes() (linkedmetaprocess.LinkedMetaProcessSystem method), 15
linkedmetaprocess (module), 13
LinkedMetaProcessSystem (class in linkedmetaprocess), 13
load_from_file() (linkedmetaprocess.LinkedMetaProcessSystem method), 15

M

MetaProcess (class in metaprocess), 11
metaprocess (module), 11
mp_data (metaprocess.MetaProcess attribute), 12

P

pad_cuts() (metaprocess.MetaProcess method), 12
pad_outputs() (metaprocess.MetaProcess method), 12
pp (metaprocess.MetaProcess attribute), 12
processes (linkedmetaprocess.LinkedMetaProcessSystem attribute), 15
product_process_dict() (linkedmetaprocess.LinkedMetaProcessSystem method), 15
products (linkedmetaprocess.LinkedMetaProcessSystem attribute), 15

R

remove_cuts_from_chain() (metaprocess.MetaProcess method), 13

remove_mp() (linkedmetaprocess.LinkedMetaProcessSystem method), 15

S

save_as_bw2_dataset() (metaprocess.MetaProcess method), 13
save_to_file() (linkedmetaprocess.LinkedMetaProcessSystem method), 15
scaling_vector_foreground_demand() (linkedmetaprocess.LinkedMetaProcessSystem method), 15

U

update() (linkedmetaprocess.LinkedMetaProcessSystem method), 15
update_name_map() (linkedmetaprocess.LinkedMetaProcessSystem method), 15
upstream_products_processes() (linkedmetaprocess.LinkedMetaProcessSystem method), 15