Brightway2-temporalis Documentation

Release 0.6

Chris Mutel

December 10, 2014

A library for the Brightway2 LCA calculation framework that allows for a specific kind of dynamic life cycle assessments.

Brightway2-temporalis is open source. Source code is available on bitbucket, and documentation is available on read the docs.

Brightway2-temporalis has the following abilities:

- Exchanges (technosphere inputs, and biosphere outputs) can be offset in time.
- Individual exchanges can be split into multiple time steps, creating a temporal distribution for each exchange.
- Inventory datasets can be given either relative or absolute dates and times.
- Characterization factors can vary as a function of time.
- Characterization factors can spread impact over time.

However, Brightway2-temporalis has the following limitations:

- Inventory datasets cannot change their inputs as a function of time. This limitation is necessary for the graph traversal to converge.
- Exchanges must be linear, as in normal matrix-based LCA.

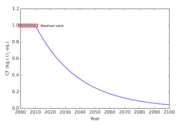
See the example ipython notebook (nbviewer, html) for a real usage example.

CHAPTER 1

Table of contents

1.1 Strategy for dynamic LCA calculations

The first step before we do anything more complicated is to take our dynamic LCIA method, which has characterization factors which can vary with time, and create a "worst case" LCIA method where the maximum possible value of each characterization factor is used:



Next, we need to reduce some of the complexity of our product systems. Inventory databases can have thousands of different process datasets, with ten of thousands of connecting links. Most of these will not be important for our specific IA method and functional unit.

We use the graph traversal algorithm, which treats the technosphere matrix as a mathematical graph, as a screening step to get a list of process datasets which *could* be important. The worst case IA method is used for LCIA calculations, because if an input is not important (in the sense of contributing to the total LCA score) applying even the highest possible characterization factors, then we can safely exclude it. The result of this step is a list of inventory datasets (nodes) and exchanges (edges) that should be further investigated.

We then start from the functional unit, and traverse this reduced supply chain graph to determine the amount of each input, and when that input occurs. The biosphere flows for each process input are also calculated. We traverse through the supply chain until either the impact of a particular input falls below a cutoff criteria (by default, 0.1% of the total possible impact), or until the maximum number of traversal steps has been reached (by default, 10.000).

The result of this second traversal is a list of biosphere flows located in time. Specifically, for each element in the list we know:

- When it occurs
- What the biosphere flow is
- The amount of the biosphere flow
- What inventory dataset caused the biosphere flow

We can then construct different timelines, both of emissions of one biosphere flow, or of total emissions. We can also apply dynamic (or static) characterization factors to create a timeline of total environmental impact over time.

1.2 Comparison with ESPA

Didier Beloin-Saint-Pierre proposed an approach called enhanced structural path analysis, which uses power series expansion and convolution to propagate relative temporal differences through the supply chain.

In the language of graph algorithm, the ESPA approach (power series expansion) is a breadth-first search, while graph traversal (at least as implemented in Brightway2-temporalis) is a depth-first search.

1.2.1 Benefits of temporal graph traversal

The first obvious benefit is that we can include both relative and absolute dates. Because we would manually traverse the supply chain graph, we can have certain activity datasets happen at absolute dates. This could be especially helpful for infrastructure built in the past.

Another advantage is that there is no fixed time steps. Each exchange has a relative time difference, but this difference can have arbitrary precision.

Finally, this approach more closely builds on the existing foundation of Brightway2, making it easier to program and test.

1.2.2 Drawbacks of temporal graph traversal

Graph traversal, like power series expansion, can only approximate the solution to a set of linear equations. An infinite number of graph traversal steps would be required to get the precise solution. However, in most cases graph traversal will converge on the precise answer relatively quickly.

1.2.3 Cut-off criteria

Suply chain graphs include loops (e.g. steel needed to generate electricity needed to make steel), and as such can be traversed without end. Cut-off criteria are needed to tell the traversal algorithm that no more work on this particular input is needed, as almost all of its impacts have already been accounted for. Similarly, power series expansion must stop after some number of calculations.

The default cutoff in Brightway2 is that inputs which, throughout their life cycle, contribute less than 0.5 percent of the total LCA score, are no longer traversed.

For temporal graph traversal, we need to be a bit more clever. First, we don't know beforehand what the total LCA score is, because the characterization factors will vary throughout time. In other words, we can't know the total LCA score before starting our calculation. However, we can estimate the *upper bound* of what that score could be, by doing a standard LCA calculation and applying the **highest** characterization factors. We can also lower the cut-off numeric criteria to 0.1 percent of the maximum possible LCA score to make sure we aren't prematurely excluding any supply chain branches.

1.3 Data formats

1.3.1 Inventory

Exchanges with temporal distributions

Both inventory dataset inputs and biosphere flows (i.e. exchanges) can be distributed in time, and can occur both before and after the inventory dataset itself. This distribution is specified in the new key `temporal distribution`:

```
"exchanges": [
    {
        "amount": 1000,
        "temporal distribution": [
            (0,  500),
            (7.5, 250),
            (15, 250)
        ]
    }
]
```

Each tuple in `temporal distribution` has the format `(relative temporal difference (in years), amount)`. Temporal differences can be positive or negative, and give the difference between when the inventory dataset and the exchange occur.

The default unit of time is years, but fractional years are allowed.

The sum of all amounts in the temporal distribution should equal the total exchange amount. This is **not** checked automatically, but can be checked using the `utils.check_temporal_distribution_totals` function.

1.3.2 Impact Assessment

Dynamic impact assessment methods

Brightway2-temporalis supports three types of characterization factors for use in dynamic LCA:

1. Static characterization factors, i.e. those which do not change over time.
2. Dynamic characterization factors, i.e. those whose value changes over time, but whose impact still occurs at the time of emission.
3. Extended dynamic characterization factors, i.e. CFs whose impact is allocated over time using something like atmospheric decay rates.

Impact assessment methods must be defined as a `DynamicIAMethod`, not a normal LCIA Method, even if all CFs are static (see *Impact Assessment methods*).

The data format for dynamic IA methods is:

```
{
    ("biosphere", "flow"): number or python_function_as_string
}
```

Note: This data format is different than the normal method data; it is a dictionary, not a list.

Static characterization factors

Static characterization factors can be defined as usual, e.g.

```
{
    ("biosphere", "n2o"): 296,
    ("biosphere", "chloroform"): 30,
}
```

Dynamic characterization factors

Dynamic characterization factors are realized with pure python functions, e.g.

```
def silly_random_cf(datetime):
    import random
    return random.random()

def increasing_co2_importance(datetime):
    """Importance of CO2 doubles every twenty years from 2010"""
    CF = 1.
    dt = arrow.get(datetime)
    cutoff = arrow.get(2010, 1, 1)
    return max(1, 2 ** ((dt - cutoff).days / 365.24 / 20) * CF)

def days_since_best_movie_evar(datetime):
    """http://en.wikipedia.org/wiki/Transformers:_Dark_of_the_Moon"""
    return (arrow.get(dt) - arrow.get(2011, 6, 23)).days
```

However, there are some things to bear in mind with dynamic characterization functions:

- Dynamic characterization functions must take a datetime as the single input, and return a single numeric characterization factor.
-
  - You will need to import whatever you need in the body of the function; don't assume anything other than the standard library is in the current namespace.
- Functions have to be defined in a slightly funny way. They should not be defined by name. Instead, they should have a name of "`%s`" that can be generated automatically and substituted by the temporalis library, i.e. `def %s(datetime)`. This dynamicity is needed to avoid name conflicts.
- These functions must be stored as **unicode strings**, not actual python code:

```
{
    ("omg", "wtf-bbq"): """def %s(datetime):
return (arrow.get(datetime) - arrow.get(2011, 6, 23)).days"""
}
```

This can be a bit confusing. See the examples for a real-world implementation.

These function strings will be executed using `exec`. Don't accept dynamic characterization function code from strange men in dark alleyways.

Extended dynamic characterization factors

Extended dynamic characterization functions don't return a single number, but rather a list of characterization factors allocated over time.

Returned CFs must be named tuples with field names `dt`, and `amount`.

```
def spread_over_a_week(datetime):
    """Spread impact over a week"""
    from datetime import timedelta
    import collections
    return_tuple = collections.namedtuple('return_tuple', ['dt', 'amount'])
    return [return_tuple(datetime + timedelta(days=x), 1 / 7.) for x in range(7)]
```

See also functions in the examples.

Aside from the return format, they are identical to normal dynamic characterization factors, and have the same restrictions.

1.4 Gotchas

1.4.1 Temporal distributions sums are not checked

The sum of all amounts in a `temporal distribution` is not checked to sum to the total `amount` by default. You can check these amounts using `utils.check_temporal_distribution_totals(my_database_name)` function.

1.4.2 Processes with specific temporal distributions could be incorrectly excluded

The initial graph traversal could exclude some nodes which have important temporal dynamics, but whose total demanded amount was small. For example, the following exchange would be excluded as having no impact, because the total amount was zero:

```
{
    "amount": 0,
    "temporal distribution": [
        (0, -1e6),
        (10, 1e6)
    ]
}
```

The best way around this software feature/bug is to create two separate sub-processes, one with the positive amounts and the other with the negative.

1.5 Technical guide

1.5.1 Impact Assessment methods

class `bw2temporalis.dynamic_ia_methods.`**`DynamicMethods`**`(`*dirpath=None*`)`

A dictionary for dynamic impact assessment method metadata. File data is saved in `dynamic-methods.json`.

class `bw2temporalis.dynamic_ia_methods.`**`DynamicIAMethod`**`(`*name*`)`

A dynamic impact assessment method. Not translated into matrices, so no `process` method.

`create_functions``(`*data=None*`)`

Take method data that defines functions in strings, and turn them into actual Python code. Returns a dictionary with flows as keys and functions as values.

`process``()`

Dynamic CFs can't be translated into a matrix, so this is a no-op.

`to_worst_case_method``(`*name*, *lower=None*, *upper=None*`)`

Create a static LCA method using the worst case for each dynamic CF function.

Default time interval over which to test for maximum CF is 2000 to 2100.

1.5.2 Dynamic Life Cycle Assessment

class `bw2temporalis.dynamic_lca.`**`DynamicLCA`**(*demand*, *worst_case_method*, *now=None*, *max_calc_number=10000.0*, *cutoff=0.001*, *log=False*, *gt_kwargs={}*)

Calculate a dynamic LCA, where processes, emissions, and CFs can vary throughout time.

1.5.3 Timeline

class `bw2temporalis.timeline.`**`Timeline`**(*data=None*)

Sum and group elements over time.

Timeline calculations produce a list of [(datetime, amount)] tuples.

`add`(*dt*, *flow*, *ds*, *amount*)

Add a new flow from a dataset at a certain time.

`timeline_for_activity`(*activity*)

Create a new Timeline for a particular activity.

`timeline_for_flow`(*flow*)

Create a new Timeline for a particular flow.