Implement a new Ensemble Analysis

The BRAPH 2 Developers

December 24, 2024

This is the developer tutorial for implementing a new ensemble analysis. In this tutorial, you will learn how to create a `*.gen.m` for a new ensemble analysis, which can then be compiled by `braph2genesis`. Here, you will use as examples the ensemble analysis `AnalyzeEnsemble_CON_BUD`, an ensemble-based graph analysis (AnalyzeEnsemble) analyzing connectivity data (CON) using binary undirected multigraphs with fixed densities (BUD).

Contents

Implementatoin of the ensemble analysis 2

Implementatoin of the ensemble analysis

You will implement in detail AnalyzeEnsemble_CON_BUD, a direct extension of AnalyzeEnsemble. An AnalyzeEnsemble_CON_BUD processes connectivity data to construct binary undirected graphs at fixed densities.

Code 1: **AnalyzeEnsemble_CON_BUD element header.** The header section of the generator code in _AnalyzeEnsemble_CON_BUD.gen.m provides the general information about the AnalyzeEnsemble_CON_BUD element.

```
%% ¡header!
AnalyzeEnsemble_CON_BUD < AnalyzeEnsemble (a, graph analysis with
     connectivity data of fixed density) is an ensemble-based graph analysis
      using connectivity data of fixed density. ①

%%% ¡description! ②
This ensemble-based graph analysis (AnalyzeEnsemble_CON_BUD) analyzes
connectivity data using binary undirected multigraphs with fixed densities.

%%% ¡seealso!
SubjectCON, MultigraphBUD

%%% ¡build! ③
1
```

① defines AnalyzeEnsemble_CON_BUD as a subclass of AnalyzeEnsemble. The moniker will be a.
② provides a description of this ensemble analysis.

③ defines the build number of the ensemble analysis element.

Code 2: **AnalyzeEnsemble_CON_BUD element prop update.** The props_update section of the generator code in _AnalyzeEnsemble_CON_BUD.gen.m updates the properties of the AnalyzeEnsemble element. This defines the core properties of the ensemble analysis.

```
%% ¡props_update!

%%% ¡prop!
ELCLASS (constant, string) is the class of the ensemble-based graph analysis
      using connectivity data of fixed density.
%%%% ¡default!
'AnalyzeEnsemble_CON_BUD'

%%% ¡prop!
NAME (constant, string) is the name of the ensemble-based graph analysis
     using connectivity data of fixed density.
%%%% ¡default!
'Connectivity Binary Undirected at fixed Density Analyze Ensemble'

%%% ¡prop!
DESCRIPTION (constant, string) is the description of the ensemble-based
     graph analysis using connectivity data of fixed density.
%%%% ¡default!
'This ensemble-based graph analysis (AnalyzeEnsemble_CON_BUD) analyzes
     connectivity data using binary undirected multigraphs with fixed
     densities.'

%%% ¡prop!
```

```
TEMPLATE (parameter, item) is the template of the ensemble-based graph analysis using connectivity data of fixed density.
%%%% ¡settings!
'AnalyzeEnsemble_CON_BUD'

%%% ¡prop!
ID (data, string) is a few-letter code for the ensemble-based graph analysis using connectivity data of fixed density.
%%%% ¡default!
'AnalyzeEnsemble_CON_BUD ID'

%%% ¡prop!
LABEL (metadata, string) is an extended label of the ensemble-based graph analysis using connectivity data of fixed density.
%%%% ¡default!
'AnalyzeEnsemble_CON_BUD label'

%%% ¡prop!
NOTES (metadata, string) are some specific notes about the ensemble-based graph analysis using connectivity data of fixed density.
%%%% ¡default!
'AnalyzeEnsemble_CON_BUD notes'

%%% ¡prop! ①
GR (data, item) is the subject group, which also defines the subject class SubjectCON.
%%%% ¡default!
Group('SUB_CLASS', 'SubjectCON')

%%% ¡prop! ②
GRAPH_TEMPLATE (parameter, item) is the graph template to set all graph and measure parameters.
%%%% ¡settings!
'MultigraphBUD'

%%% ¡prop! ③
G_DICT (result, idict) is the graph (MultigraphBUD) ensemble obtained from this analysis.
%%%% ¡settings!
'MultigraphBUD'
%%%% ¡calculate!
g_dict = IndexedDictionary('IT_CLASS', 'MultigraphBUD');
gr = a.get('GR');
densities = a.get('DENSITIES'); ④

for i = 1:1:gr.get('SUB_DICT').get('LENGTH') ⑤
  sub = gr.get('SUB_DICT').get('IT', i);
    g = MultigraphBUD( ...⑥
        'ID', ['graph ' sub.get('ID')], ...
        'B', sub.getCallback('CON'), ...
        'DENSITIES', densities, ... ⑦
        'LAYERLABELS', cellfun(@(x) [num2str(x) '%'], num2cell(densities), 'UniformOutput', false), ...
        'NODELABELS', a.get('GR').get('SUB_DICT').get('IT', 1).get('BA').get('BR_DICT').get('KEYS') ...
        );
    g_dict.get('ADD', g) ⑧
end
```

① defines the property GR, which contains the subjects data using SubjectCON element, which are the subjects to be analyzed.

② Specifies the GRAPH_TEMPLATE to define parameters such as DENSITIES, SEMIPOSITIVIZE_RULE, and STANDARDIZE_RULE. These settings are applied to all graphs in ④. Here, the graph element used is MultigraphBUD. ③ creates G_DICT, a graph dictionary that contains instances of MultigraphBUD. These instances are derived from the subjects defined in ①.

④ retrieves the densities defined in the new properties below, which is used to configure the MultigraphBUD instances for the analysis. ⑤, ⑥, ⑦, and ⑧ collectively build the graph dictionary (G_DICT). This process begins by iterating over each subject in GR, constructing an instance of MultigraphBUD for each subject based on their respective data, applying the specified DENSITIES parameter, and finally adding the created MultigraphBUD instances into the dictionary.

```
if ~isa(a.get('GRAPH_TEMPLATE'), 'NoValue')
    for i = 1:1:g_dict.get('LENGTH')
        g_dict.get('IT', i).set('TEMPLATE', a.get('GRAPH_TEMPLATE')) ⑨
    end
end

value = g_dict;

%%% ¡prop!
ME_DICT (result, idict) contains the calculated measures of the graph ensemble.
```

⑨ ensures that all `MultigraphBUD` instances in the dictionary are updated with the pre-defined parameters from the graph template specified in ②, if explicitly set by the user during initialization of `AnalyzeEnsemble_CON_BUD`.

Code 3: **AnalyzeEnsemble_CON_BUD element props.** The `props` section of the generator code in `_AnalyzeEnsemble_CON_BUD.gen.m` defines the properties to be used in `AnalyzeEnsemble_CON_BUD`.

```
%% ¡props!

%%% ¡prop!
DENSITIES (parameter, rvector) is the vector of densities.
%%%% ¡default!
[1:1:10]
%%%% ¡gui! ①
pr = PanelPropRVectorSmart('EL', a, 'PROP', AnalyzeEnsemble_CON_BUD.DENSITIES, ...
    'MIN', 0, 'MAX', 100, ...
    'DEFAULT', AnalyzeEnsemble_CON_BUD.getPropDefault('DENSITIES'), ...
    varargin{:});
%%%% ¡postset!
a.memorize('GRAPH_TEMPLATE').set('DENSITIES', a.getCallback('DENSITIES'));
```

① `PanelPropRVectorSmart` plots the panel for a row vector with an edit field. Smart means that (almost) any MatLab expression leading to a correct row vector can be introduced in the edit field. Also, the value of the vector can be limited between some MIN and MAX.

Code 4: **AnalyzeEnsemble_CON_BUD element tests.** The `tests` section in the element generator `_AnalyzeEnsemble_CON_BUD.gen.m`. A general test should be prepared for using the example script where the ensemble analysis is used. The test should also at least verify in some simple cases that the GUI function is working properly.

```

%% ¡tests!

%%% ¡excluded_props! ①
[AnalyzeEnsemble_CON_BUD.TEMPLATE AnalyzeEnsemble_CON_BUD.GRAPH_TEMPLATE]

%%% ¡test! ②
%%%% ¡name!
Example
%%%% ¡probability! ③
.01
.01
%%%% ¡code!
create_data_CON_XLS() % only creates files if the example folder doesn't already exist

example_CON_BUD

%%% ¡test! ④
```

① List of properties that are excluded from testing.

② Tests the functionality of `AnalyzeEnsemble_CON_BUD` using an example script.

③ assigns a low test execution probability.

④ Tests the GUI functionality of `AnalyzeEnsemble_CON_BUD`.

```
%%%% ¡name!
GUI - Analysis
%%%% ¡probability!
.01
%%%% ¡code!
im_ba = ImporterBrainAtlasXLS(’FILE’, ’desikan_atlas.xlsx’);
ba = im_ba.get(’BA’); ⑤

gr = Group(’SUB_CLASS’, ’SubjectCON’, ’SUB_DICT’, IndexedDictionary(’IT_CLASS’, ’SubjectCON’)); ⑥
for i = 1:1:50
    sub = SubjectCON( ...
        ’ID’, [’SUB CON ’ int2str(i)], ...
        ’LABEL’, [’Subejct CON ’ int2str(i)], ...
        ’NOTES’, [’Notes on subject CON ’ int2str(i)], ...
        ’BA’, ba, ...
        ’CON’, rand(ba.get(’BR_DICT’).get(’LENGTH’)) ...
        );
    sub.memorize(’VOI_DICT’).get(’ADD’, VOINumeric(’ID’, ’Age’, ’V’, 100 * rand()))
    sub.memorize(’VOI_DICT’).get(’ADD’, VOICategoric(’ID’, ’Sex’, ’CATEGORIES’, {’Female’, ’Male’}, ’V’, randi(2, 1)))
    gr.get(’SUB_DICT’).get(’ADD’, sub)
end

a = AnalyzeEnsemble_CON_BUD(’GR’, gr, ’DENSITIES’, 5:5:20); ⑦

gui = GUIElement(’PE’, a, ’CLOSEREQ’, false);⑧
gui.get(’DRAW’) ⑨
gui.get(’SHOW’) ⑩

gui.get(’CLOSE’) ⑪

%%% ¡test! ⑫
%%%% ¡name!
GUI - Comparison
%%%% ¡probability!
.01
%%%% ¡code!
im_ba = ImporterBrainAtlasXLS(’FILE’, ’desikan_atlas.xlsx’);
ba = im_ba.get(’BA’);

gr1 = Group(’SUB_CLASS’, ’SubjectCON’, ’SUB_DICT’, IndexedDictionary(’IT_CLASS’, ’SubjectCON’));
for i = 1:1:50
    sub = SubjectCON( ...
        ’ID’, [’SUB CON ’ int2str(i)], ...
        ’LABEL’, [’Subejct CON ’ int2str(i)], ...
        ’NOTES’, [’Notes on subject CON ’ int2str(i)], ...
        ’BA’, ba, ...
        ’CON’, rand(ba.get(’BR_DICT’).get(’LENGTH’)) ...
        );
    sub.memorize(’VOI_DICT’).get(’ADD’, VOINumeric(’ID’, ’Age’, ’V’, 100 * rand()))
    sub.memorize(’VOI_DICT’).get(’ADD’, VOICategoric(’ID’, ’Sex’, ’CATEGORIES’, {’Female’, ’Male’}, ’V’, randi(2, 1)))
    gr1.get(’SUB_DICT’).get(’ADD’, sub)
end
```

⑤ and ⑥ define the necessary objects required to initialize an instance of `AnalyzeEnsemble_CON_BUD`.

⑦ Initializes an `AnalyzeEnsemble_CON_BUD` instance using the specified `gr` (group) and `densities`.

⑧, ⑨, and ⑩ test the process of creating a GUI for `AnalyzeEnsemble_CON_BUD`, drawing it, and showing it on the screen.

⑪ tests the process of closing the shown GUI.

⑫ tests the GUI functionality for another use case of `AnalyzeEnsemble_CON_BUD`.

```

gr2 = Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('IT_CLASS', 'SubjectCON'));
for i = 1:1:50
    sub = SubjectCON( ...
        'ID', ['SUB CON ' int2str(i)], ...
        'LABEL', ['Subejct CON ' int2str(i)], ...
        'NOTES', ['Notes on subject CON ' int2str(i)], ...
        'BA', ba, ...
        'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
        );
    sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 * rand()))
    sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', 'CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
    gr2.get('SUB_DICT').get('ADD', sub)
end

a1 = AnalyzeEnsemble_CON_BUD('GR', gr1, 'DENSITIES', 5:5:20); ⑬
a2 = AnalyzeEnsemble\_CON\_BUD('GR', gr2, 'TEMPLATE', a1);⑭

c = CompareEnsemble( ...⑮
    'P', 10, ...
    'A1', a1, ...
    'A2', a2, ...
    'WAITBAR', true, ...
    'VERBOSE', false, ...
    'MEMORIZE', true ...
    );

gui = GUIElement('PE', c, 'CLOSEREQ', false);  ⑯
gui.get('DRAW') ⑰
gui.get('SHOW') ⑱

gui.get('CLOSE') ⑲
```

⑬ Similar to the previous test, this initializes the first `AnalyzeEnsemble_CON_BUD` with the specified `gr` and densities.

⑭ Initializes the second `AnalyzeEnsemble_CON_BUD` using the first `AnalyzeEnsemble_CON_BUD` instance as a template. This setup allows the second instance to have its own `gr` data while applying the same parameters, specifically the densities.

⑮ creates a `CompareEnsemble` instance with the defined `AnalyzeEnsemble_CON_BUD` instances.

⑯, ⑰, ⑱, and ⑲ test creating, drawing, showing, and closing the GUI of the `CompareEnsemble`.