# *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 at fixed densities (BUD).

## *Contents*

## *Implementation 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.

### *Basic properties*

This section focuses on implementing the basic properties required to define `AnalyzeEnsemble_CON_BUD`, including its class, name, and associated metadata.

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.

```
%% iheader!
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)

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

%%% iseealso!
SubjectCON, MultigraphBUD

%%% ibuild! (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: **Basic properties of AnalyzeEnsemble_CON_BUD.** This section of the generator code in `_AnalyzeEnsemble_CON_BUD.gen.m` updates the basic properties required to describe the `AnalyzeEnsemble_CON_BUD` element, including its class, name, description, and other metadata.

```
%% iprops_update!

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

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

%%% iprop!
```

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

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

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

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

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

### *Functionality-focused properties*

This section details the implementation of functionality-focused properties that enable `AnalyzeEnsemble_CON_BUD` to perform graph analysis directly.

Code 3: **Implementation properties of AnalyzeEnsemble_CON_BUD.** This section of the generator code in `_AnalyzeEnsemble_CON_BUD.gen.m` updates the properties to be used, including `GR` for defining subjects' data, `GRAPH_TEMPLATE` for specifying graph type and parameters, and `G_DICT` for managing graph instances across subjects.

```

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

%%% iprop! (2)
GRAPH_TEMPLATE (parameter, item) is the graph template to set all graph and
     measure parameters.
%%%% isettings!
'MultigraphBUD'
```

(1) defines the property `GR`, which stores the subjects using `SubjectCON` element, containing the subjects' data 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 (3). Here, the graph element used is `MultigraphBUD`.

```
11
12 %%% iprop! (3)
13 G_DICT (result, idict) is the graph (MultigraphBUD) ensemble obtained from
       this analysis.
14 %%%% isettings!
15 'MultigraphBUD'
16 %%%% icalculate!
17 g_dict = IndexedDictionary('IT_CLASS', 'MultigraphBUD');
18 gr = a.get('GR');
19 densities = a.get('DENSITIES'); (4)
20
21 for i = 1:1:gr.get('SUB_DICT').get('LENGTH') (5)
22   sub = gr.get('SUB_DICT').get('IT', i);
23     g = MultigraphBUD( ...(6)
24         'ID', ['graph ' sub.get('ID')], ...
25         'B', sub.getCallback('CON'), ...
26         'DENSITIES', densities, ... (7)
27         'LAYERLABELS', cellfun(@(x) [num2str(x) '%'], num2cell(densities), '
       UniformOutput', false), ...
28         'NODELABELS', a.get('GR').get('SUB_DICT').get('IT', 1).get('BA').get
       ('BR_DICT').get('KEYS') ...
29         );
30     g_dict.get('ADD', g) (8)
31 end
32
33 if ~isa(a.get('GRAPH_TEMPLATE'), 'NoValue')
34     for i = 1:1:g_dict.get('LENGTH')
35         g_dict.get('IT', i).set('TEMPLATE', a.get('GRAPH_TEMPLATE'))  (9)
36     end
37 end
38
39 value = g_dict;
40
41 %%% iprop!
42 ME_DICT (result, idict) contains the calculated measures of the graph
       ensemble.
```

(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.

(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 4: **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`.

```
1 %% iprops!
2
3 %%% iprop! (1)
4 DENSITIES (parameter, rvector) is the vector of densities.
5 %%%% idefault!
6 [1:1:10]
7 %%%% igui! (2)
8 pr = PanelPropRVectorSmart('EL', a, 'PROP', AnalyzeEnsemble_CON_BUD.
       DENSITIES, ...
9     'MIN', 0, 'MAX', 100, ...
10    'DEFAULT', AnalyzeEnsemble_CON_BUD.getPropDefault('DENSITIES'), ...
11    varargin{:});
12 %%%% ipostset! (3)
13 a.memorize('GRAPH_TEMPLATE').set('DENSITIES', a.getCallback('DENSITIES'));
```

(1) defines the densities for binarizing the connectivity matrix in a binary undirected multigraph, used in (7) of Code 3

(2) `PanelPropRVectorSmart` plots a GUI row vector panel for defining densities, supporting MATLAB expressions and limiting values between MIN and MAX.

(3) handles postprocessing after `DENSITIES` is set, memorizing a `GRAPH_TEMPLATE` with the defined `DENSITIES`, applied later in (9) of Code 3.

*Verification through testing*

This section validates `AnalyzeEnsemble_CON_BUD` by implementing tests to confirm its functionality via example scripts and ensure GUI integration.

Code 5: **AnalyzeEnsemble_CON_BUD element tests.** The `tests` section in the element generator `_AnalyzeEnsemble_CON_BUD.gen.m` includes logic tests, which verify correct functionality using example scripts and simulated datasets, and integration tests, which ensure the instance operation of the direct GUI and associated GUIs.

```
%% itests!

%%% iexcluded_props! (1)
[AnalyzeEnsemble_CON_BUD.TEMPLATE AnalyzeEnsemble_CON_BUD.GRAPH_TEMPLATE]

%%% itest! (2)
%%%% iname!
Example
%%%% iprobability! (3)
.01
.01
%%%% icode!
create_data_CON_XLS() % only creates files if the example folder doesn't
     already exist

example_CON_BUD

%%% itest! (4)
%%%% iname!
GUI - Analysis
%%%% iprobability!
.01
%%%% icode!
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', ['Subject 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)
```

(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 direct GUI functionality of `AnalyzeEnsemble_CON_BUD`.

(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`.

```
42
43 gui = GUIElement('PE', a, 'CLOSEREQ', false);(8)
44 gui.get('DRAW') (9)
45 gui.get('SHOW') (10)
46
47 gui.get('CLOSE') (11)
48
49 %%% itest! (12)
50 %%%% iname!
51 GUI - Comparison
52 %%%% iprobability!
53 .01
54 %%%% icode!
55 im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
56 ba = im_ba.get('BA');
57
58 gr1 = Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectCON'));
59 for i = 1:1:50
60     sub = SubjectCON( ...
61         'ID', ['SUB CON ' int2str(i)], ...
62         'LABEL', ['Subject CON ' int2str(i)], ...
63         'NOTES', ['Notes on subject CON ' int2str(i)], ...
64         'BA', ba, ...
65         'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
66         );
67     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
68     sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
69     gr1.get('SUB_DICT').get('ADD', sub)
70 end
71
72 gr2 = Group('SUB_CLASS', 'SubjectCON', 'SUB_DICT', IndexedDictionary('
     IT_CLASS', 'SubjectCON'));
73 for i = 1:1:50
74     sub = SubjectCON( ...
75         'ID', ['SUB CON ' int2str(i)], ...
76         'LABEL', ['Subject CON ' int2str(i)], ...
77         'NOTES', ['Notes on subject CON ' int2str(i)], ...
78         'BA', ba, ...
79         'CON', rand(ba.get('BR_DICT').get('LENGTH')) ...
80         );
81     sub.memorize('VOI_DICT').get('ADD', VOINumeric('ID', 'Age', 'V', 100 *
     rand()))
82     sub.memorize('VOI_DICT').get('ADD', VOICategoric('ID', 'Sex', '
     CATEGORIES', {'Female', 'Male'}, 'V', randi(2, 1)))
83     gr2.get('SUB_DICT').get('ADD', sub)
84 end
85
86 a1 = AnalyzeEnsemble_CON_BUD('GR', gr1, 'DENSITIES', 5:5:20); (13)
87 a2 = AnalyzeEnsemble\_CON\_BUD('GR', gr2, 'TEMPLATE', a1);(14)
88
89 c = CompareEnsemble( ...(15)
90     'P', 10, ...
91     'A1', a1, ...
92     'A2', a2, ...
```

(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 associated GUI functionality of `AnalyzeEnsemble_CON_BUD`.

(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.

```
93      'WAITBAR', true, ...
94      'VERBOSE', false, ...
95      'MEMORIZE', true ...
96      );
97
98  gui = GUIElement('PE', c, 'CLOSEREQ', false);   (16)
99  gui.get('DRAW')  (17)
100 gui.get('SHOW')  (18)
101
102 gui.get('CLOSE')  (19)
```

(16), (17), (18), and (19) test creating, drawing, showing, and closing the GUI of the `CompareEnsemble`, which is an associated GUI of `AnalyzeEnsemble_CON_BUD`