

# *Implement a new Graph*

*The BRAPH 2 Developers*

*August 31, 2023*

This is the developer tutorial for implementing a new graph. In this Tutorial, we will explain how to create the generator file `*.gen.m` for a new graph, which can then be compiled by `braph2genesis`. All graphs are (direct or indirect) extensions of the base element `Graph`. Here, we will use as examples the graphs `GraphBD` (Binary Directed graph), `MultilayerWU` (Weighted Undirected multilayer graph), `MultiplexBUT` (Binary Undirected multiplex at fixed Thresholds), and `OrdMxBUT` (Binary Undirected ordinal multiplex with fixed Thresholds).

## *Contents*

<i>Implementation of Unilayer Graphs</i>	3
<i>Unilayer Binary Directed Graph (GraphBD)</i>	3
<i>Implementation of Multilayer Graphs</i>	8
<i>Weigthed Directed Multilayer Graph (MultilayerWD )</i>	8
<i>Binary Undirected Multilayer Graph with fixed Thresholds (MultiplexBUT)</i>	13
<i>Binary Undirected Ordinal Multiplex Graph with fixed Thresholds (OrdMxBUT)</i>	18

## Implementation of Unilayer Graphs

### Unilayer Binary Directed Graph (GraphBD)

We will start by implementing in detail GraphBD, which is a direct extension of Graph. A unilayer graph is constituted by nodes connected by edges, where the edges are directed and they can be either 0 (absence of connection) or 1 (existence of connection).

**Code 1: GraphBD element header.** The header section of the generator code for `_GraphBD.gen.m` provides the general information about the GraphBD element.

---

```

1 %% iheader!
2 GraphBD < Graph (g, binary directed graph) is a binary directed graph. ①
3
4 %%% idescription!
5 In a binary directed (BD) graph, the edges are directed and they can be
   either 0 (absence of connection) or 1 (existence of connection).
```

---

① defines GraphBD as a subclass of Graph. The moniker will be g.

**Code 2: GraphBD element prop update.** The `props_update` section of the generator code for `GraphBD.gen.m` updates the properties of the Graph element. This defines the core properties of the graph.

---

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the binary directed graph.
5 %%% idefault!
6 'GraphBD'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the binary directed
   graph.
10 %%% idefault!
11 'In a binary directed (BD) graph, the edges are directed and they can be
   either 0 (absence of connection) or 1 (existence of connection).'
```

---

```

12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the binary directed graph.
15
16 %%% iprop!
17 ID (data, string) is a few-letter code of the binary directed graph.
18 %%% idefault!
19 'GraphBD ID'
20
21 %%% iprop!
22 LABEL (metadata, string) is an extended label of the binary directed graph.
23 %%% idefault!
24 'GraphBD label'
25
26 %%% iprop!
27 NOTES (metadata, string) are some specific notes about the binary directed
   graph.
28 %%% idefault!
29 'GraphBD notes'
30
```

```

31 %% iprop! ①
32 GRAPH_TYPE (constant, scalar) returns the graph type __Graph.GRAPH__.
33 %%%% idefault!
34 Graph.GRAPH
35
36 %% iprop! ②
37 CONNECTIVITY_TYPE (query, smatrix) returns the connectivity type __Graph.
    BINARY__.
38 %%%% idefault!
39 value = Graph.BINARY;
40
41 %% iprop! ③
42 DIRECTIONALITY_TYPE (query, smatrix) returns the directionality type
    __Graph.DIRECTED__.
43 %%%% idefault!
44 value = Graph.DIRECTED;
45
46 %% iprop! ④
47 SELFCONNECTIVITY_TYPE (query, smatrix) returns the self-connectivity type
    __Graph.NONSELFCONNECTED__.
48 %%%% idefault!
49 value = Graph.NONSELFCONNECTED;
50
51 %% iprop! ⑤
52 NEGATIVITY_TYPE (query, smatrix) returns the negativity type __Graph.
    NONNEGATIVE__.
53 %%%% idefault!
54 value = Graph.NONNEGATIVE;
55
56 %% iprop! ⑥
57 A (result, cell) is the binary adjacency matrix of the binary directed graph
    .
58 %%%% icalculate!
59 B = g.get('B'); ⑦
60
61 B = dediagonalize(B); ⑧
62 B = semipositivize(B, 'SemipositivizeRule', g.get('SEMIPOSITIVIZE_RULE'));
    ⑨
63 B = binarize(B); ⑩
64
65 A = {B}; ⑪
66 value = A; ⑫
67
68 %%%% igui! ⑬
69 pr = PanelPropCell('EL', g, 'PROP', GraphBD.A, ...
70 'TABLE_HEIGHT', s(40), ...
71 'XSLIDERSHOW', false, ...
72 'YSLIDERSHOW', false, ...
73 'ROWNAME', g.getCallback('ANODELABELS'), ...
74 'COLUMNNAME', g.getCallback('ANODELABELS'));
75
76 %% iprop! ⑭
77 COMPATIBLE_MEASURES (constant, classlist) is the list of compatible measures
    .
78 %%%% idefault!
79 getCompatibleMeasures('GraphBD')

```

① defines the *graph type*: Graph.GRAPH (single layer), Graph.MULTIGRAPH (multiple unconnected layers), Graph.MULTILAYER (multiple layers), Graph.ORDERED\_MULTILAYER (multiple layers with subsequent layers) Graph.MULTIPLEX (multilayer with connections between corresponding nodes), and Graph.ORDERED\_MULTIPLEX (multilayer with connections between corresponding nodes in subsequent layers).

② defines the *graph connectivity*: Graph.BINARY (0 or 1) or Graph.WEIGHTED.

③ defines the *edge directionality*: Graph.DIRECTED or Graph.UNDIRECTED.

④ defines the *graph self-connectivity*: Graph.NONSELFCONNECTED or Graph.SELFCONNECTED.

⑤ defines the *graph negativity*: Graph.NONNEGATIVE or Graph.NEGATIVE.

⑥ The property A contains the supra-adjacency matrix of the graph, which is calculated by the code under icalculate!.

⑦ retrieves the adjacency matrix of the graph B, defined in the new properties below.

⑧, ⑨, and ⑩ condition the adjacency matrix removing the diagonal elements, making it semidefinite positive, and binarizing it. A list of useful functions is: diagonalize (removes the off-diagonal), dediagonalize (removes the diagonal), binarize (binarizes with threshold=0), semipositivize (removes negative weights), standardize (normalizes between 0 and 1) or symmetrize (symmetrizes the matrix). Use the MatLab help to see additional functionalities.

⑪ preallocates the adjacency matrix to be calculated.

⑫ returns the calculated graph A assigning it to the output variable value.

⑬ employs the property panel PanelPropCell to be employed to visualize A, setting also its properties.

⑭ determines the list of compatible figures.

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

---

```

1 %% iprops!
2
3 %%% iprop! ①
4 B (data, smatrix) is the input graph adjacency matrix.
5 %%% igui! ②
6 pr = PanelPropMatrix('EL', g, 'PROP', GraphBD.B, ...
7   'TABLE_HEIGHT', s(40), ...
8   'ROWNAME', g.getCallback('ANODELABELS'), ...
9   'COLUMNNAME', g.getCallback('ANODELABELS'), ...
10   varargin{:});
11
12 %%% iprop! ③
13 SEMIPOSITIVIZE_RULE (parameter, option) determines how to remove the
   negative edges.
14 %%% isettings!
15 {'zero', 'absolute'}

```

---

① contains the input adjacency matrix B, which is typically weighted and directed.

② defines the property panel PanelPropMatrix to plot this property with a table.

③ defines the semi-positivation rule (i.e., how to remove the negative edges) to be used when generating the adjacency matrix A from the input property B. The admissible options are: 'zero' (default, convert negative values to zeros) or 'absolute' (convert negative values to absolute value).

Code 4: **GraphBD element tests.** The tests section from the element generator `_GraphBD.gen.m`. A general test should be prepared to test the properties of the graph when it is empty and full. Furthermore, additional tests should be prepared for the rules defined (one test per rule).

```

1 %% itests!
2
3 %% iexcluded_props! (1)
4 [GraphBD.PFGA GraphBD.PFGH]
5
6 %% itest!
7 %%% iname!
8 Constructor - Empty (2)
9 %%% iprobability! (3)
10 .01
11 %%% icode!
12 B = []; (4)
13 g = GraphBD('B', B); (5)
14
15 g.get('A_CHECK'); (6)
16
17 A = {binarize(semipositivize(dediagonalize(B)))}; (7)
18 assert(isequal(g.get('A'), A), ... (8)
19 [BRAPH2.STR ':GraphBD:' BRAPH2.FAIL_TEST], ...
20 'GraphBD is not constructing well.')
21
22 %% itest!
23 %%% iname!
24 Constructor - Full (9)
25 %%% iprobability!
26 .01
27 %%% icode!
28 B = randn(randi(10)); (10)
29 g = GraphBD('B', B);
30
31 g.get('A_CHECK')
32
33 A = {binarize(semipositivize(dediagonalize(B)))};
34 assert(isequal(g.get('A'), A), ...
35 [BRAPH2.STR ':GraphBD:' BRAPH2.FAIL_TEST], ...
36 'GraphBD is not constructing well.')
37
38 %% itest!
39 %%% iname!
40 Semipositivize Rules (11)
41 %%% iprobability!
42 .01 (3)
43 %%% icode!
44 B = [ (12)
45     -2 -1 0 1 2
46     -1 0 1 2 -2
47     0 1 2 -2 -1
48     1 2 -2 -1 0
49     2 -2 -1 0 1
50 ];

```

(1) List of properties that are excluded from testing.

(2) checks that an empty GraphBD graph is constructing well.

(3) assigns a low test execution probability.

(4) initializes an empty input adjacency matrix B.

(5) constructs the GraphBD graph from the initialized B.

(6) performs the corresponding checks for the format of the adjacency matrix A: GRAPH\_TYPE, CONNECTIVITY\_TYPE, DIRECTIONALITY\_TYPE, SELFCONNECTIVITY\_TYPE, and NEGATIVITY\_TYPE.

(7) calculates the value of the graph by apply the corresponding properties function.

(8) tests that the value of generated graph calculated by applying the properties functions coincides with the expected value.

(9) checks that a full GraphBD graph is constructing well.

(10) generates a random input adjacency matrix B.

(11) checks the SEMIPOSITIVIZE\_RULE on the GraphBD graph.

(12) generates an input adjacency matrix with negative weights.

```

51
52 g0 = GraphBD('B', B); (13)
53 A0 = {[ (14)
54     0 0 0 1 1
55     0 0 1 1 0
56     0 1 0 0 0
57     1 1 0 0 0
58     1 0 0 0 0
59     ]};
60 assert(isequal(g0.get('A'), A0), ...
61     [BRAPH2.STR ':GraphBD:' BRAPH2.FAIL_TEST], ...
62     'GraphBD is not constructing well.')
63
64 g_zero = GraphBD('B', B, 'SEMIPOSITIVIZE_RULE', 'zero'); (15)
65 A_zero = {[
66     0 0 0 1 1
67     0 0 1 1 0
68     0 1 0 0 0
69     1 1 0 0 0
70     1 0 0 0 0
71     ]};
72 assert(isequal(g_zero.get('A'), A_zero), ...
73     [BRAPH2.STR ':GraphBD:' BRAPH2.FAIL_TEST], ...
74     'GraphBD is not constructing well.')
75
76 g_absolute = GraphBD('B', B, 'SEMIPOSITIVIZE_RULE', 'absolute'); (16)
77 A_absolute = {[
78     0 1 0 1 1
79     1 0 1 1 1
80     0 1 0 1 1
81     1 1 1 0 0
82     1 1 1 0 0
83     ]};
84 assert(isequal(g_absolute.get('A'), A_absolute), ...
85     [BRAPH2.STR ':GraphBD:' BRAPH2.FAIL_TEST], ...
86     'GraphBD is not constructing well.')
87 ...

```

(13) constructs the GraphBD graph from the initialized B with default RULE for SEMIPOSITIVIZE\_RULE.

(14) provides the expected value of A calculated by external means.

(15) constructs the GraphBD graph from the initialized B with RULE = 'zero' for SEMIPOSITIVIZE\_RULE.

(16) constructs the GraphBD graph from the initialized B with RULE = 'absolute' for SEMIPOSITIVIZE\_RULE.

## Implementation of Multilayer Graphs

### Weighted Directed Multilayer Graph (MultilayerWD)

We can now use GraphBD as the basis to implement the MultilayerWD graph. The parts of the code that are modified are highlighted. A multilayer graph allows connections between any nodes across the multiple layers, where all layers are interconnected following a categorical fashion.

**Code 5: MultilayerWD element header.** The header section of generator code for `_MultilayerWD.gen.m` provides the general information about the MultilayerWD element. [← Code 1](#)

---

```

1 %% iheader!
2 MultilayerWD < Graph (g, multilayer weighted directed graph) is a multilayer
  weighted directed graph.
3
4 %%% idescription!
5 In a multilayer weighted directed (WD) graph, layers could have different
  number of nodes with within-layer weighted directed edges, associated
  with a real number between 0 and 1 and indicating the strength of the
  connection. The connectivity matrices are symmetric (within layer). All
  node connections are allowed between layers.
```

---

**Code 6: MultilayerWD element prop update.** The `props_update` section of generator code for `_MultilayerWD.gen.m` updates the properties of MultilayerWD. [← Code 2](#)

---

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the multilayer weighted directed
  graph.
5 %%% idefault!
6 'MultilayerWD'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the multilayer weighted
  directed graph.
10 %%% idefault!
11 'In a multilayer weighted directed (WD) graph, layers could have different
  number of nodes with within-layer weighted directed edges, associated
  with a realnumber between 0 and 1 and indicating the strength of the
  connection. The connectivity matrices are symmetric (within layer). All
  node connections are allowed between layers.'
```

---

```

12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the multilayer weighted
  directed graph.
15
16 %%% iprop!
17 ID (data, string) is a few-letter code of the multilayer weighted directed
  graph.
18 %%% idefault!
19 'MultilayerWD ID'
20
```

```

21 %%% iprop!
22 LABEL (metadata, string) is an extended label of the multilayer weighted
    directed graph.
23 %%% idefault!
24 'MultilayerWD label'
25
26 %%% iprop!
27 NOTES (metadata, string) are some specific notes about the multilayer
    weighted directed graph.
28 %%% idefault!
29 'MultilayerWD notes'
30
31 %%% iprop!
32 GRAPH_TYPE (constant, scalar) returns the graph type __Graph.MULTILAYER__.
33 %%% idefault!
34 Graph.MULTILAYER
35
36 %%% iprop!
37 CONNECTIVITY_TYPE (query, smatrix) returns the connectivity type __Graph.
    WEIGHTED__ * ones(layernumber).
38 %%% icalculate!
39 if isempty(varargin)
40     layernumber = 1;
41 else
42     layernumber = varargin{1};
43 end
44 value = Graph.WEIGHTED * ones(layernumber);
45
46 %%% iprop!
47 DIRECTIONALITY_TYPE (query, smatrix) returns the directionality type __Graph.
    .DIRECTED__ * ones(layernumber).
48 %%% icalculate!
49 if isempty(varargin)
50     layernumber = 1;
51 else
52     layernumber = varargin{1};
53 end
54 value = Graph.DIRECTED * ones(layernumber);
55
56 %%% iprop!
57 SELFCONNECTIVITY_TYPE (query, smatrix) returns the self-connectivity type
    __Graph.NONSELFCONNECTED__ on the diagonal and __Graph.SELFCONNECTED__
    off diagonal.
58 %%% icalculate!
59 if isempty(varargin)
60     layernumber = 1;
61 else
62     layernumber = varargin{1};
63 end
64 value = Graph.SELFCONNECTED * ones(layernumber);
65 value(1:layernumber+1:end) = Graph.NONSELFCONNECTED;
66
67 %%% iprop!
68 NEGATIVITY_TYPE (query, smatrix) returns the negativity type __Graph.
    NONNEGATIVE__ * ones(layernumber).
69 %%% icalculate!
70 if isempty(varargin)
71     layernumber = 1;
72 else
73     layernumber = varargin{1};
74 end

```



```

75 value = Graph.NONNEGATIVE * ones(layernumber);
76
77 %%% iprop!
78 A (result, cell) is the cell containing the within-layer weighted adjacency
79 matrices of the multilayer weighted directed graph and the connections
80 between layers.
81
82 %%% icalculate!
83 B = g.get('B');
84 L = length(B);
85 A = cell(L, L);
86 for i = 1:L (1)
87     M = dedagonalize(B{i,i});
88     M = semipositivize(M, 'SemipositivizeRule', g.get('SEMIPOSITIVIZE_RULE'));
89     M = standardize(M, 'StandardizeRule', g.get('STANDARDIZE_RULE'));
90     A(i, i) = {M};
91     if ~isempty(A{i, i})
92         for j = i+1:L
93             M = semipositivize(B{i,j}, 'SemipositivizeRule', g.get('
SEMIPPOSITIVIZE_RULE'));
94             M = standardize(M, 'StandardizeRule', g.get('STANDARDIZE_RULE'));
95             A(i, j) = {M};
96             M = semipositivize(B{j,i}, 'SemipositivizeRule', g.get('
SEMIPPOSITIVIZE_RULE'));
97             M = standardize(M, 'StandardizeRule', g.get('STANDARDIZE_RULE'));
98             A(j, i) = {M};
99         end
100     end
101 end
102 value = A;
103 %%% igui!
104 pr = PanelPropCell('EL', g, 'PROP', MultilayerWD.A, ...
105     'TABLE_HEIGHT', s(40), ...
106     'XYSLIDERLOCK', true, ...
107     'XSLIDERSHOW', false, ...
108     'YSLIDERSHOW', true, ...
109     'YSLIDERLABELS', g.getCallback('ALAYERLABELS'), ...
110     'YSLIDERWIDTH', s(5), ...
111     'ROWNAME', g.getCallback('ANODELABELS'), ...
112     'COLUMNNAME', g.getCallback('ANODELABELS'), ...
113     varargin{:});
114
115 %%% iprop!
116 PARTITIONS (result, rvector) returns the number of layers in the partitions
117 of the graph.
118 %%% icalculate!
119 value = ones(1, g.get('LAYERNUMBER'));
120
121 %%% iprop! (2)
122 ALAYERLABELS (query, stringlist) returns the layer labels to be used by the
123 slider.
124 %%% icalculate!
125 alayerlabels = g.get('LAYERLABELS'); (3)
126 if isempty(alayerlabels) && ~isa(g.get('A'), 'NoValue') % ensures that it's
127 not unnecessarily calculated
128     alayerlabels = cellfun(@num2str, num2cell([1:L:g.get('LAYERNUMBER')]), '
uniformoutput', false); (4)
129 end
130 value = alayerlabels;
131

```

(1) For each layer in MultilayerWD graph, the corresponding functions are applied as in the notes (8), (9), and (10) of Code 2.

(2) These are some properties of graph adjacency matrix A that can be used in the gui to make the visualization user friendly. The list of properties that can be used are: ALAYERTICKS (to set ticks for each layer according to the layer number), ALAYERLABELS (to set labels for each layer), and ANODELABELS (to set the node labels for each layer).

(3) returns the labels of the graph layers provided by the user.

(4) constructs the labels of the layers based on the number of the layer (in case no layer labels were provided by the user).

```

129 %%% iprop!
130 COMPATIBLE_MEASURES (constant, classlist) is the list of compatible measures
131 %%% idefault!
132 getCompatibleMeasures('MultilayerWD')

```

---

**Code 7: MultilayerWD element props.** The props section of generator code for MultilayerWD.gen.m defines the properties to be used in MultilayerWD. ← [Code 3](#)

---

```

1 %% iprops!
2
3 %%% iprop!
4 B (data, cell) is the input cell containing the multilayer adjacency
   matrices.
5 %%% idefault!
6 {} []; {} []
7 %%% igui! ①
8 pr = PanelPropCell('EL', g, 'PROP', MultilayerWD.B, ...
9   'TABLE_HEIGHT', s(40), ...
10  'XSLIDERSHOW', true, ...
11  'XSLIDERLABELS', g.get('LAYERLABELS'), ...
12  'XSLIDERHEIGHT', s(3.5), ...
13  'YSLIDERSHOW', false, ...
14  'ROWNAME', g.getCallback('ANODELABELS'), ...
15  'COLUMNNAME', g.getCallback('ANODELABELS'), ...
16  varargin{:});
17
18
19 %%% iprop!
20 SEMIPOSITIVIZE_RULE (parameter, option) determines how to remove the
   negative edges.
21 %%% isettings!
22 {'zero', 'absolute'}
23
24 %%% iprop! ②
25 STANDARDIZE_RULE (parameter, option) determines how to normalize the weights
   between 0 and 1.
26 %%% isettings!
27 {'threshold' 'range'}

```

---

① Same as in note ② of Code 3.

② Same as in note ③ of Code 3.

**Code 8: MultilayerWD element tests.** The tests section from the element generator \_MultilayerWD.gen.m. ← [Code 4](#)

---

```

1 %% itests!
2
3 %%% iexcluded_props!
4 [MultilayerWD.PFGA MultilayerWD.PFGH]
5
6 %%% itest!
7 %%% iname!
8 Constructor - Full
9 %%% iprobability!
10 .01
11 %%% icode!
12 B1 = rand(randi(10));
13 B2 = rand(randi(10));
14 B3 = rand(randi(10));
15 B12 = rand(size(B1, 1), size(B2, 2));

```

```

16 B13 = rand(size(B1, 1),size(B3, 2));
17 B23 = rand(size(B2, 1),size(B3, 2));
18 B21 = rand(size(B2, 1),size(B1, 2));
19 B31 = rand(size(B3, 1),size(B1, 2));
20 B32 = rand(size(B3, 1),size(B2, 2));
21 B = {
22     B1                B12                B13
23     B21                B2                B23
24     B31                B32                B3
25 };
26 g = MultilayerWD('B', B);
27 g.get('A_CHECK')
28 A1 = standardize(semipositivize(dediagonalize(B1)));
29 A2 = standardize(semipositivize(dediagonalize(B2)));
30 A3 = standardize(semipositivize(dediagonalize(B3)));
31 A12 = standardize(semipositivize(B12));
32 A13 = standardize(semipositivize(B13));
33 A23 = standardize(semipositivize(B23));
34 A21 = standardize(semipositivize(B21));
35 A31 = standardize(semipositivize(B31));
36 A32 = standardize(semipositivize(B32));
37 B{1,1} = A1;
38 B{2,2} = A2;
39 B{3,3} = A3;
40 B{1,2} = A12;
41 B{1,3} = A13;
42 B{2,3} = A23;
43 B{2,1} = A21;
44 B{3,1} = A31;
45 B{3,2} = A32;
46 A = B;
47 assert(isequal(g.get('A'), A), ...
48     [BRAPH2.STR ': MultilayerWD: ' BRAPH2.FAIL_TEST], ...
49     'MultilayerWD is not constructing well.')

```

---

### *Binary Undirected Multilayer Graph with fixed Thresholds (MultiplexBUT)*

Now we implement the MultiplexBUT graph based on previous codes GraphBD and MultilayerWD, again highlighting the differences. A multiplex graph is a type of multilayer graph where only interlayer edges are allowed between homologous nodes. In this case, the layers follow a categorical architecture, which means that all layers are interconnected.

**Code 9: MultiplexBUT element header.** The header section of generator code for `_MultiplexBUT.gen.m` provides the general information about the MultiplexBUT element. ← [Code 1](#)

---

```

1 %% iheader!
2 MultiplexBUT < MultiplexWU (g, binary undirected multiplex with fixed
    thresholds) is a binary undirected multiplex with fixed thresholds. ①
3
4 %%% idescription!
5 In a binary undirected multiplex with fixed thresholds (BUT), the layers are
    those of binary undirected (BU) multiplex graphs derived from the same
    weighted supra-connectivity matrices binarized at different thresholds
    .The supra-connectivity matrix has a number of partitions equal to the
    number of thresholds.
```

---

① MultiplexBUT is a child of MultiplexWU, which in turn derives from Graph.

**Code 10: MultiplexBUT element prop update.** The `props_update` section of generator code for `_MultiplexBUT.gen.m` updates the properties of MultiplexBUT. ← [Code 2](#)

---

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the binary undirected multiplex with
    fixed thresholds.
5 %%% idefault!
6 'MultiplexBUT'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the binary undirected
    multiplex with fixed thresholds.
10 %%% idefault!
11 'In a binary undirected multiplex with fixed thresholds (BUT), the layers
    are those of binary undirected (BU) multiplex graphs derived from the
    same weighted supra-connectivity matrices binarized at different
    thresholds. The supra-connectivity matrix has a number of partitions
    equal to the number of thresholds.'
```

---

```

12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the binary undirected
    multiplex with fixed thresholds.
15
16 %%% iprop!
17 ID (data, string) is a few-letter code of the binary undirected multiplex
    with fixed thresholds.
18 %%% idefault!
19 'MultiplexBUT ID'
20
21 %%% iprop!
```

```

22 LABEL (metadata, string) is an extended label of the binary undirected
    multiplex with fixed thresholds.
23 %%%% idefault!
24 'MultiplexBUT label'
25
26 %%% iprop!
27 NOTES (metadata, string) are some specific notes about the binary undirected
    multiplex with fixed thresholds.
28 %%%% idefault!
29 'MultiplexBUT notes'
30
31 %%% iprop!
32 GRAPH_TYPE (constant, scalar) returns the graph type __Graph.MULTIPLEX__.
33 %%%% idefault!
34 Graph.MULTIPLEX
35
36 %%% iprop!
37 CONNECTIVITY_TYPE (query, smatrix) returns the connectivity type __Graph.
    BINARY__ * ones(layernumber).
38 %%%% icalculate!
39 if isempty(varargin)
40     layernumber = 1;
41 else
42     layernumber = varargin{1};
43 end
44 value = Graph.BINARY * ones(layernumber);
45
46 %%% iprop!
47 DIRECTIONALITY_TYPE (query, smatrix) returns the directionality type __Graph.
    UNDIRECTED__ * ones(layernumber).
48 %%%% icalculate!
49 if isempty(varargin)
50     layernumber = 1;
51 else
52     layernumber = varargin{1};
53 end
54 value = Graph.UNDIRECTED * ones(layernumber);
55
56 %%% iprop!
57 SELFCONNECTIVITY_TYPE (query, smatrix) returns the self-connectivity type
    __Graph.NONSELFCONNECTED__ on the diagonal and __Graph.SELFCONNECTED__
    off diagonal.
58 %%%% icalculate!
59 if isempty(varargin)
60     layernumber = 1;
61 else
62     layernumber = varargin{1};
63 end
64 value = Graph.SELFCONNECTED * ones(layernumber);
65 value(1:layernumber+1:end) = Graph.NONSELFCONNECTED;
66
67 %%% iprop!
68 NEGATIVITY_TYPE (query, smatrix) returns the negativity type __Graph.
    NONNEGATIVE__ * ones(layernumber).
69 %%%% icalculate!
70 if isempty(varargin)
71     layernumber = 1;
72 else
73     layernumber = varargin{1};
74 end
75 value = Graph.NONNEGATIVE * ones(layernumber);

```

```

76
77 %%% iprop!
78 A (result, cell) is the cell containing multiplex binary adjacency matrices
    of the binary undirected multiplex.
79
80 %%% icalculate!
81 A_WU = calculateValue@MultiplexWU(g, prop); ①
82
83 thresholds = g.get('THRESHOLDS'); ②
84 L = length(A_WU); % number of layers ③
85 A = cell(length(thresholds) * L); ④
86
87 if L > 0 && ~isempty(cell2mat(A_WU))
88     A(:, :) = {eye(length(A_WU{1, 1}))};
89     for i = 1:length(thresholds) ⑤
90         threshold = thresholds(i);
91         layer = 1;
92         for j = (i - 1) * L + 1:L * L ⑥
93             A{j, j} = dedagonalize(binthesize(A_WU{layer, layer}, 'threshold',
94                 threshold)); ⑦
95             layer = layer + 1;
96         end
97     end
98
99 value = A;
100
101 %%% igui! ⑧
102 pr = PanelPropCell('EL', g, 'PROP', MultiplexBUT.A, ...
103     'TABLE_HEIGHT', s(40), ...
104     'XYSLIDERLOCK', true, ...
105     'XSLIDERSHOW', false, ...
106     'YSLIDERSHOW', true, ...
107     'YSLIDERLABELS', g.getCallback('ALAYERLABELS'), ...
108     'YSLIDERWIDTH', s(5), ...
109     'ROWNAME', g.getCallback('ANODELABELS'), ...
110     'COLUMNNAME', g.getCallback('ANODELABELS'), ...
111     varargin{:});
112
113 %%% iprop!
114 PARTITIONS (result, rvector) returns the number of layers in the partitions
    of the graph.
115 %%% icalculate!
116 l = g.get('LAYERNUMBER');
117 thresholds = g.get('THRESHOLDS');
118 value = ones(1, length(thresholds)) * l / length(thresholds);
119
120 %%% iprop!
121 ALAYERLABELS (query, stringlist) returns the layer labels to be used by the
    slider.
122 %%% icalculate!
123 alayerlabels = g.get('LAYERLABELS');
124 if ~isa(g.get('A'), 'NoValue') && length(alayerlabels) ~= g.get('
    LAYERNUMBER') % ensures that it's not unnecessarily calculated
125     thresholds = cellfun(@num2str, num2cell(g.get('THRESHOLDS')), '
    uniformoutput', false);
126
127 if length(alayerlabels) == length(g.get('B'))
128     blayerlabels = alayerlabels;

```

- ① calculates the graph MultiplexWU calling its parent MultiplexWU.
- ② gets the thresholds to be applied to A\_WU.
- ③ gets the number of layers in graph A\_WU.
- ④ The new MultiplexBUT graph will have L layers for each threshold applied.
- ⑤ iterates over all the thresholds to be applied.
- ⑥ iterates over all the layers in A\_WU.
- ⑦ binarizes the present layer of the A\_WU graph according to the present threshold.

- ⑧ Same as in note ② of Code 2.

```

129 else % includes isempty(layerlabels)
130     blayerlabels = cellfun(@num2str, num2cell([1:1:length(g.get('B'))]), '
        uniformoutput', false);
131 end
132
133 alayerlabels = {};
134 for i = 1:1:length(thresholds) ⑨
135     for j = 1:1:length(blayerlabels)
136         alayerlabels = [alayerlabels, [blayerlabels{j} '|' thresholds{i}]];
137     end
138 end
139 end
140 value = alayerlabels;
141
142 %% iprop!
143 COMPATIBLE_MEASURES (constant, classlist) is the list of compatible measures
144
145 %%%% idefault!
146 getCompatibleMeasures('MultiplexBUT')

```

⑨ sets the labels of the layers considering the thresholds and the number of layers in each multiplex graph for each threshold

**Code 11: MultiplexBUT element props.** The props section of generator code for MultiplexBUT.gen.m defines the properties to be used in MultiplexBUT. ← [Code 3](#)

```

1 %% iprops!
2
3 %% iprop!
4 THRESHOLDS (parameter, rvector) is the vector of thresholds.
5 %%%% igui! ①
6 pr = PanelPropRVectorSmart('EL', g, 'PROP', MultiplexBUT.THRESHOLDS, ...
7     'MAX', 1, ...
8     'MIN', -1, ...
9     varargin{:});

```

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 12: MultiplexBUT element tests.** The tests section from the element generator \_MultiplexBUT.gen.m. ← [Code 4](#)

```

1 %% itests!
2
3 %% itest!
4 %%%% iname!
5 Constructor - Full
6 %%%% iprobability!
7 .01
8 %%%% icode!
9 B1 = [
10     0 .1 .2 .3 .4
11     .1 0 .1 .2 .3
12     .2 .1 0 .1 .2
13     .3 .2 .1 0 .1
14     .4 .3 .2 .1 0
15 ];
16 B = {B1, B1, B1}; ①
17 thresholds = [0 .1 .2 .3 .4]; ②
18 g = MultiplexBUT('B', B, 'THRESHOLDS', thresholds);
19
20 g.get('A-CHECK')
21

```

① creates an example of the necessary input adjacency matrices.

② defines the thresholds.

```
22 A = g.get('A');
23 for i = 1:1:length(B) * length(thresholds)
24     for j = 1:1:length(B) * length(thresholds)
25         if i == j
26             threshold = thresholds(floor((i - 1) / length(B)) + 1);
27             assert(isequal(A{i, i}, binarize(B1, 'threshold', threshold)), ...
28                 [BRAPH2.STR ':MultiplexBUT:' BRAPH2.FAIL_TEST], ...
29                 'MultiplexBUT is not constructing well.')
30         else
31             assert(isequal(A{i, j}, eye(length(B1))), ...
32                 [BRAPH2.STR ':MultiplexBUT:' BRAPH2.FAIL_TEST], ...
33                 'MultiplexBUT is not constructing well.')
34         end
35     end
36 end
```

---



### *Binary Undirected Ordinal Multiplex Graph with fixed Thresholds (OrdMxBUT)*

Finally, we implement the OrdMxBUT graph based on previous codes GraphBD, MultilayerWD and MultiplexBUT, again highlighting the differences. An ordered multiplex is a type of multiplex graph that consists of a sequence of layers with ordinal edges between corresponding nodes in subsequent layers.

**Code 13: OrdMxBUT element header.** The header section of generator code for `_OrdMxBUT.gen.m` provides the general information about the OrdMxBUT element. ← [Code 1](#)

---

```

1 %% iheader!
2 OrdMxBUT < OrdMxWU (g, ordinal multiplex binary undirected with fixed
    thresholds) is a binary undirected ordinal multiplex with fixed
    thresholds. ①
3
4 %%% idescription!
5 In a binary undirected ordinal multiplex with fixed thresholds (BUT), all
    the layers consist of binary undirected (BU) multiplex graphs derived
    from the same weighted supra-connectivity matrices binarized at
    different thresholds. The supra-connectivity matrix has a number of
    partitions equal to the number of thresholds. The layers are connected
    in an ordinal fashion, i.e., only consecutive layers are connected.

```

---

① OrdMxBUT is a child of OrdMxWU, which in turn derives from Graph.

**Code 14: OrdMxBUT element prop update.** The `props_update` section of generator code for `_OrdMxBUT.gen.m` updates the properties of OrdMxBUT. ← [Code 10](#)

---

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the binary undirected ordinal
    multiplex with fixed thresholds.s.
5 %%% idefault!
6 'OrdMxBUT'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the binary undirected
    ordinal multiplex with fixed thresholds..
10 %%% idefault!
11 'In a binary undirected ordinal multiplex with fixed thresholds (BUT), all
    the layers consist of binary undirected (BU) multiplex graphs derived
    from the same weighted supra-connectivity matrices binarized at
    different thresholds. The supra-connectivity matrix has a number of
    partitions equal to the number of thresholds. The layers are
    connected in an ordinal fashion, i.e., only consecutive layers are
    connected.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the binary undirected ordinal
    multiplex with fixed thresholds.
15
16 %%% iprop!
17 ID (data, string) is a few-letter code of the binary undirected ordinal
    multiplex with fixed thresholds.

```

---

```

18 %%%% idefault!
19 'OrdMxBUT ID'
20
21 %%% iprop!
22 LABEL (metadata, string) is an extended label of the binary undirected
    ordinal multiplex with fixed thresholds.
23 %%%% idefault!
24 'OrdMxBUT label'
25
26 %%% iprop!
27 NOTES (metadata, string) are some specific notes about the binary undirected
    ordinal multiplex with fixed thresholds.
28 %%%% idefault!
29 'OrdMxBUT notes'
30
31 %%% iprop!
32 GRAPH_TYPE (constant, scalar) returns the graph type __Graph.
    ORDERED_MULTIPLEX__.
33 %%%% idefault!
34 Graph.ORDERED_MULTIPLEX
35
36 %%% iprop!
37 CONNECTIVITY_TYPE (query, smatrix) returns the connectivity type __Graph.
    BINARY__ * ones(layernumber).
38 %%%% icalculate!
39 if isempty(varargin)
40     layernumber = 1;
41 else
42     layernumber = varargin{1};
43 end
44 value = Graph.BINARY * ones(layernumber);
45
46 %%% iprop!
47 DIRECTIONALITY_TYPE (query, smatrix) returns the directionality type __Graph.
    UNDIRECTED__ * ones(layernumber).
48 %%%% icalculate!
49 if isempty(varargin)
50     layernumber = 1;
51 else
52     layernumber = varargin{1};
53 end
54 value = Graph.UNDIRECTED * ones(layernumber);
55
56 %%% iprop!
57 SELFCONNECTIVITY_TYPE (query, smatrix) returns the self-connectivity type
    __Graph.NONSELFCONNECTED__ on the diagonal and __Graph.SELFCONNECTED__
    off diagonal.
58 %%%% icalculate!
59 if isempty(varargin)
60     layernumber = 1;
61 else
62     layernumber = varargin{1};
63 end
64 value = Graph.SELFCONNECTED * ones(layernumber);
65 value(1:layernumber+1:end) = Graph.NONSELFCONNECTED;
66
67 %%% iprop!
68 NEGATIVITY_TYPE (query, smatrix) returns the negativity type __Graph.
    NONNEGATIVE__ * ones(layernumber).
69 %%%% icalculate!
70 if isempty(varargin)

```

```

71 layernumber = 1;
72 else
73 layernumber = varargin{1};
74 end
75 value = Graph.NONNEGATIVE * ones(layernumber);
76
77 %%% iprop!
78 A (result, cell) is the cell containing binary supra-adjacency matrix of the
    binary undirected multiplex with fixed thresholds (BUT).
79
80 %%% icalculate!
81 A_WU = calculateValue@OrdMxWU(g, prop); ①
82
83 thresholds = g.get('THRESHOLDS'); ②
84 L = length(A_WU); % number of layers
85 A = cell(length(thresholds)*L);
86
87 if L > 0 && ~isempty(cell2mat(A_WU))
88 A(:, :) = {zeros(length(A_WU{1, 1}))};
89 for i = 1:length(thresholds) ③
90 threshold = thresholds(i);
91 layer = 1;
92 for j = (i - 1) * L + 1:L * L ④
93 for k = (i - 1) * L + 1:L * L
94 if j == k ⑤
95 A{j, j} = dedagonalize(binarize(A_WU{layer, layer}, 'threshold',
    threshold));
96 elseif (j-k)==1 || (k-j)==1 ⑥
97 A{j, k} = {eye(length(A{1, 1}))};
98 else ⑦
99 A{j, k} = {zeros(length(A{1, 1}))};
100 end
101 end
102 layer = layer + 1;
103 end
104 end
105 end
106
107 value = A;
108
109 %%% igui!
110 pr = PanelPropCell('EL', g, 'PROP', OrdMxBUT.A, ...
111 'TABLE_HEIGHT', s(40), ...
112 'XYSLIDERLOCK', true, ...
113 'XSLIDERSHOW', false, ...
114 'YSLIDERSHOW', true, ...
115 'YSLIDERLABELS', g.getCallback('ALAYERLABELS'), ...
116 'YSLIDERWIDTH', s(5), ...
117 'ROWNAME', g.getCallback('ANODELABELS'), ...
118 'COLUMNNAME', g.getCallback('ANODELABELS'), ...
119 varargin{:});
120
121 %%% iprop!
122 PARTITIONS (result, rvector) returns the number of layers in the partitions
    of the graph.
123 %%% icalculate!
124 l = g.get('LAYERNUMBER');
125 thresholds = g.get('THRESHOLDS');
126 value = ones(1, length(thresholds)) * l / length(thresholds);

```

① calculates the graph OrdMxWU calling the parent OrdMxWU.

② Same as in notes ②, ③, and ④ of Code 10..

③ constructs an ordinal multiplex binary undirected graph for each threshold.

④ loops over the layers of A\_Wu for each threshold.

⑤ sets the layers constructed by binarizing A\_Wu according to the present threshold on the diagonal of the supra-adjacency matrix.

⑥ connects consecutive layers.

⑦ does NOT connect NON-consecutive layers.

```

127
128 %% iprop!
129 ALAYERLABELS (query, stringlist) returns the layer labels to be used by the
    slider.
130 %%% icalculate!
131 alayerlabels = g.get('LAYERLABELS');
132 if ~isa(g.get('A'), 'NoValue') && length(alayerlabels) ~= g.get('
    LAYERNUMBER') % ensures that it's not unnecessarily calculated
133     thresholds = cellfun(@num2str, num2cell(g.get('THRESHOLDS')), '
        uniformoutput', false);
134
135 if length(alayerlabels) == length(g.get('B'))
136     blayerlabels = alayerlabels;
137 else % includes isempty(layerlabels)
138     blayerlabels = cellfun(@num2str, num2cell([1:1:length(g.get('B'))]), '
        uniformoutput', false);
139 end
140
141 alayerlabels = {};
142 for i = 1:1:length(thresholds)
143     for j = 1:1:length(blayerlabels)
144         alayerlabels = [alayerlabels, [blayerlabels{j} '|' thresholds{i}]];
145     end
146 end
147 end
148 value = alayerlabels;
149
150 %% iprop!
151 COMPATIBLE_MEASURES (constant, classlist) is the list of compatible measures
    .
152 %%% idefault!
153 getCompatibleMeasures('OrdMxBUT')

```

---

**Code 15: OrdMxBUT element props.** The props section of generator code for `OrdMxBUT.gen.m` defines the properties to be used in `MultiplexBUT`. ← [Code 11](#)

---

```

1 %% iprops!
2
3 %% iprop!
4 THRESHOLDS (parameter, rvector) is the vector of thresholds.
5 %%% igrui!
6 pr = PanelPropRVectorSmart('EL', g, 'PROP', OrdMxBUT.THRESHOLDS, ...
7     'MAX', 1, ...
8     'MIN', -1, ...
9     varargin{:});

```

---

**Code 16: OrdMxBUT element tests.** The tests section from the element generator `_OrdMxBUT.gen.m`. ← [Code 12](#)

---

```

1 %% itests!
2
3 %% iexcluded_props!
4 [OrdMxBUT.PFGA OrdMxBUT.PFGH]
5
6 %% itest!
7 %%% iname!
8 Constructor - Full
9 %%% iprobability!

```

```

10 .01
11 %%%% icode!
12 B1 = [
13     0 .1 .2 .3 .4
14     .1 0 .1 .2 .3
15     .2 .1 0 .1 .2
16     .3 .2 .1 0 .1
17     .4 .3 .2 .1 0
18 ];
19 B = {B1, B1, B1};
20 thresholds = [0 .1 .2 .3 .4];
21 g = OrdMxBUT('B', B, 'THRESHOLDS', thresholds);
22
23 g.get('A_CHECK')
24
25 A = g.get('A');
26 for i = 1:length(thresholds)
27     threshold = thresholds(i);
28     for j = (i - 1) * length(B) + 1:i * length(B)
29         for k = (i - 1) * length(B) + 1:i * length(B)
30             if j == k
31                 assert(isequal(A{j, j}, binarize(B1, 'threshold', threshold)), ...
32                     [BRAPH2.STR ':OrdMxBUT:' BRAPH2.FAIL_TEST], ...
33                     'OrdMxBUT is not constructing well.')
34             elseif (j-k)==1 || (k-j)==1
35                 assert(isequal(A{j, k}, eye(length(B1))), ...
36                     [BRAPH2.STR ':OrdMxBUT:' BRAPH2.FAIL_TEST], ...
37                     'OrdMxBUT is not constructing well.')
38             else
39                 assert(isequal(A{j, k}, zeros(length(B1))), ...
40                     [BRAPH2.STR ':OrdMxBUT:' BRAPH2.FAIL_TEST], ...
41                     'OrdMxBUT is not constructing well.')
42             end
43         end
44     end
45 end

```

---