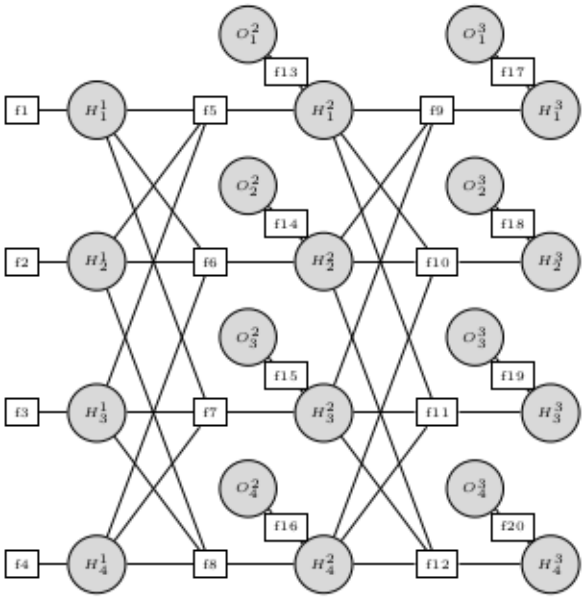


GMtoolbox

Inference in Graphical Models

Reference manual

Version 1.0.1 - August 2019

Marie-Josée Cros
Nathalie Peyrard
Régis Sabbadin

Contents

1 General description

The goal of GMtoolbox is to provide probability estimation by inference in probabilistic Graphical Models (GM) [1] for non computer science specialists.

The toolbox allows to:

- represent a graphical model as a factor graph,
- infer marginals based on apriori knowledge or observation.

The inference relies on the Generalized Belief Propagation algorithm parent-to-child [3].
To be sure of pertinence, we checked that problems of reasonable size were tractable and that execution time was not too long comparing with libDAI [2].

For functions gm_plot_fg, gm_plot_rg, gm_separate_fg and util/setCountingNumbers a version of Matlab higher than R2015b is required (because they use the graph and digraph classes).

2 Graphical model representation

2.1 Mathematical definition of a graphical model

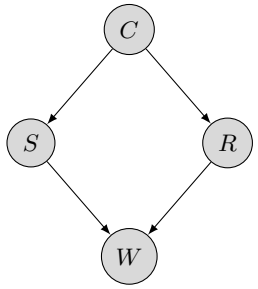
Definition from [3].
Let $X = \{X_1, \ldots, X_N\}$ be a set of N random variables taking value in a finite set. We denote by x_i a realization of variable X_i and by x a realization of the random vector X. The joint law $p(x) =_{def} P(X = x)$ is that of a graphical model if can be factorized as a product of functions, each involving only a subset of all the variables:

$$p(x) = \frac{1}{Z} \prod_{a \in \{A,B,\ldots,M\}} f_a(x_a), \tag{2.1}$$

where a is an index labeling M functions $f_A, f_B \ldots, f_M$ and the scope of function f_a is x_a, a subset of of $\{x_1, \ldots, x_N\}$. We will often identify a with the indices of the variables in x_a.
We suppose that $f_a(x_a)$ is a non negative and finite function. Z is the normalizing constant of p. If the graphical model is a Bayesian network, $Z = 1$ and the functions f_a are conditional probabilities.

A classical example of Bayesian network is the Sprinkler model. It models the relations between state of grass: wet/not (variable W), use of sprinkler: on/off (variable S), rain : yes/no (variable R) and cloudy sky : yes/no (variable C). The joint law is $p(C, S, R, W) = p(C)p(S \mid C)p(R \mid C)p(W \mid S, R)$.

p(C)	
C=true	0.5
C=false	0.5



p(S|C)	S=true)	S=false
C=true	0.5	0.5
C=false	0.9	0.1

p(R|C)	R=true	R=false
C=true	0.8	0.2
C=false	0.2	0.8

p(W|S,R)	W=true	W=false
S=true, R=true	1	0
S=true, R=false	0.9	0.1
S=false, R=true	0.9	0.1
S=false, R=false	0.01	0.99

The GMtoolbox provides tools to compute marginals of the distribution p (also called beliefs). For instance, we can use it to compute the probability distribution of variable X_i, defined by (taking $i = 1$)

$$p_1(X_1 = x_1^*) = \sum_{x_2} \sum_{x_3} \ldots \sum_{x_n} p(x_1^*, x_2, \ldots, x_n).$$

More generally, we can also use the toolbox to compute the joint law of a subset of variables $X_S =$

$\{x_i\}_{i\in S}$, with $S \subset \{1,\dots,n\}$:

$$p(X_S = x_S^*) = \sum_{x_{\bar{S}}} p(x_{\bar{S}}, x_S^*),$$

with $\bar{S} = \{1,\dots,n\} \setminus S$ The algorithm for computing these marginals relies on the factor graph representation of a graphical model, and on the notion of region graph.

2.2 Description of a factor graph

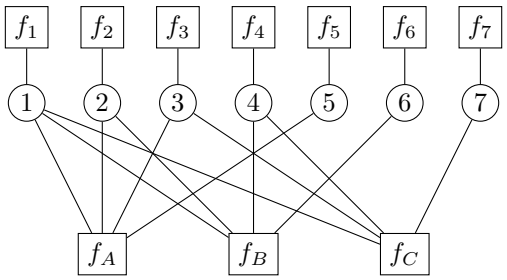
2.2.1 Definition

Definition from [3].
It is possible to associate to the factorized expression (2.1) a factor graph which represents graphically which variables are in the scope of which function. It is a bipartite graph, composed of *variable nodes* and *factor nodes*. There is one variable node per variable X_i and one factor node per function f_a. Edges are only between a variable and a factor node. There exists an edge between variable node i and factor node a if and only if x_i is in the scope of f_a.

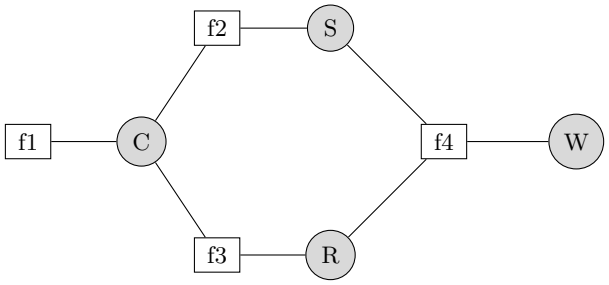
For example (see figure 13 of [3]), here is a joint law of $N = 7$ random variables, factorized into $M = 9$ factors.

$$p(x) = \frac{1}{Z}\left(\prod_{i=1}^{7} f_i(x_i)\right) f_A(x_1,x_2,x_3,x_5) f_B(x_1,x_2,x_4,x_6) f_C(x_1,x_3,x_4,x_7) \tag{2.2}$$

Here is the associated factor graph.



And the factor graph of the Sprinkler Bayesian network is composed of 4 factors f1(C), f2(C,S), f3(C,R), f4(S,R,W).



2.2.2 Matlab representation

A factor graph may be composed of more than one connex component, or can become disconnected when evidences are taken into account. Each connex component is seen a a sub factor graph of the main one, and the Matlab structure stores all the sub factor graphs in a meta structure. More precisely,a **factor graph** is represented in a structure, called `fg`.
Let nv be the number of variables, then the element of the structure are the followings:

fg.Card	Cardinality of variables array (1 x nv) of integer upper than or equal to 2
fg.Name	Names of variables (optional) cell array (1 x nv) of string
fg.sfg	All sub factor graphs cell array (1 x ...) of sub factor graph structure
fg.evidence	observation (optional) array (1 x nv) of integer

If variable 1 is of cardinality 2, possible states are 1 and 2. That is first possible state of variable is 1.

Note that is the field evidence exists, the field sfg defining sub factor graphs must be compatible with evidences. The function gm_include_evidence has this role to update sub factor graphs (and related factors) to take into account evidences.

The factor graph fg is composed of several **sub factor graphs** represented in a structure called sfg. Let nv' ($nv' < nv$) be the number of variables and nf the number of factors of a sub factor graph. The element of the structure are the followings:

sfg.Vcode	Labels of variables in fg vector (1 x nv') of integer
sfg.F	Factors cell array (1 x nf) of factors
sfg.BG	Utilitary and redundant field: compact format to express variables of each factor logical matrix (nf x nv') BG(i,j) = $true$ when factor i includes variable j

A **factor** with nq ($nq \leq nv'$) variables is represented in a structure, called factor:

factor.V	variables array (1 x nq) of integer
factor.T	table of a factor , two possible types - an array of nq dimensions: dimension d has the cardinality of the corresponding variable; - a function handler

Example
Here is the factor graph of the classic toy example Sprinkler (see 3.1 for a description).

```
function fg = gm_example_Sprinkler()
   % Define sprinkler toy example factor graph.
   % Variable state:  true/wet (coded 1) or false/dry (coded 2).
   fg.Card = [2 2 2 2]; % cardinalities of variables
   fg.Name = {'Cloudy', 'Sprinkler', 'Rain', 'Wet'}; % names of variables
   % The model is composed on a single sub-graph fg
   sfg.Vcode = [1 2 3 4];
   F1.V = [1];     % factor 1
   F1.T = @() [ 0.5; 0.5 ];     % function handler
   F2.V  = [1 2];   % factor 2
   F2.T = @() [ 0.5, 0.5; 0.9, 0.1];     % function handler
   F3.V  = [1 3];   % factor 3
   F3.T = @() [ 0.8, 0.2; 0.2 0.8];     % function handler
   F4.V  = [2 3 4]; % factor 4
   F4.T = @f4;     % function handler
   sfg.F ={F1, F2, F3, F4}; % four factors
   sfg.BG = setBG(4, sfg.F); % matrix factors x variables
   fg.sfg{1}=sfg;
```

The function setBG computes automatically the element sfg.BG from the number of factors and their definition. It is defined in the toolbox (util/setBG.m).

Tables of factors are defined using function handlers. For factor F4, the function is defined explicitly in a function called cpt.

```
function cpt = f4()
   cpt(1,1,:) = [ 1 0]; % Sprinkler, Rain
   cpt(1,2,:) = [ 0.9 0.1]; % Sprinkler, no Rain
   cpt(2,1,:) = [ 0.9 0.1]; % no Sprinkler, Rain
   cpt(2,2,:) = [ 0.01 0.99]; % no Sprinkler, no Rain
end
```

Using function handler can be useful for factor graph with several factors sharing the same table, as it can be the case in Dynamic Bayesian Networks (DBN) or in Markov Random Fields (MRF).

Since tables of factors may be defined either directly in an array or using a function handler, to see a table, it is necessary to know its type. This can be done as follow.

```
>> fg = gm_example_Sprinkler()
fg =
  struct with fields:
    Card: [2 2 2 2]
    Name: {'Cloudy'  'Sprinkler'  'Rain'  'Wet'}
     sfg: {[1×1 struct]}
>> if  isa(fg.sfg{1}.F{4}.T, 'function_handle')
      fg.sfg{1}.F{4}.T()
   else
      fg.sfg{1}.F{4}.T
   end
ans(:,:,1) =
    1.0000    0.9000
    0.9000    0.0100
ans(:,:,2) =
         0    0.1000
    0.1000    0.9900
```

Suppose now that it is observed that there is no rain and no use of sprinkler that is the state of variables 2 and 3 is 1. Evidence is then coded [0 1 1 0]. Unknown state of variable 1 and 4 is coded 0. When this evidence is included, the obtained factor graph is composed of 2 disconnected sub factor

graphs, each composed of one variable.

```
>> fg=gm_example_Sprinkler;
>> evidence = [0 1 1 0];
>> fge= gm_include_evidence(fg, evidence)
fge =
  struct with fields:
        Card: [2 2 2 2]
        Name: {'Cloudy'  'Sprinkler'  'Rain'  'Wet'}
         sfg: {[1×1 struct]  [1×1 struct]}
    evidence: [0 1 1 0]
% explore the second sub factor graph
>> fge.sfg{2}
ans =
  struct with fields:
    Vcode: 4
        F: {[1×1 struct]}
       BG: 1
% explore its factor
>> fge.sfg{2}.F{1}
ans =
  struct with fields:
    V: 1
    T: @()T
>> fge.sfg{2}.F{1}.T()
ans =
     1
     0
```

The second sub factor graph concerns variable 4 (see fge.sfg2.Vcode).
Looking at its factor, it concerns the first variable of the sub factor graph (see fge.sfg2.F1.V) . The associated table indicates that for sure grass is dry (see T(1) with T=fge.sfg2.F1.T()).

2.3 Description of a region graph

2.3.1 Definition

Definition from [3].

A *region* R is a set of variable nodes V_R and a set of factor nodes F_R such that if a factor node f_a belongs to F_R, all variable nodes linked to f_a by an edge in the factor graph must belong to V_R. On contrary, if all the variables nodes linked to a factor are in V_R, that factor do not need to be in F_R. F_R can be empty.
A *region graph* is a labeled directed (acyclic) graph $G =< V, E, L >$ which nodes are regions, and whose edges satisfy some properties. If I is the set of indices of the factor graph (variable nodes and factor nodes together), then a node $v \in V$ is labeled with a subset of I corresponding to the indices of variable and factor nodes in the region attached to v. We denote by $l(v)$ the label of node v. Then the set of edges E of a region graph satisfies the following condition: A directed edge (or arc) e may exist pointing from vertex v_p to vertex v_c if $l(v_c)$ is a subset of $l(v_p)$. All other arcs are forbidden. Therefore, by construction the region graph is acyclic.

In practice, in order to use the region graph concept to build approximate inference method, it is required that every factor and variable node of the factor graph is contained in at least one region of the region graph. Another condition is also required to ensure consistency of the beliefs computed by GBP: the regions containing a particular variable node i form a connected sub-graph of the region graph. This ensures that we can compute $p(x_i)$ by marginalizing the belief of any region that contains i and we will obtain the same result.

Several methods exist to build a valid region graph for GBP. The toolbox proposes three methods called: JT, BETHE, Cluster Variation Method (CVM).

To complete the definition of a region graph we defined the associated counting numbers, even if the Parent-to-Child algorithm does not use these numbers to build the message passing rules. however this numbers are used to check the validity of a region graph: they must sum to 1 (used in gm_check_rg function).

2.3.2 Matlab representation

A **region graph**, called rg is a set of sub region graphs, one for each sub factor graph of the graphical model. It is represented as a cell array (1 x ...), each cell containing the structure of a sub region graph, called srg. Let nr be the number of regions of the sub region graph, the structure has the following elements:

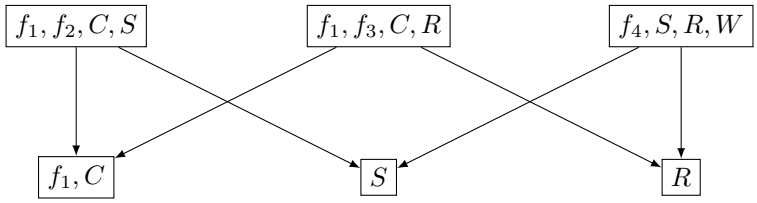
srg.Vr	variables in each region logical array (nr x nv) Vr(i,j) = $true$ when region i includes variable j
srg.Fr	factors in each region logical array (nr x nf) Fr(i,j) = $true$ when region i includes factor j
srg.Gr	adjacency matrix of regions logical array (nr x nr) Gr(i,j) = $true$ when region $i \rightarrow$ region j
srg.cr	counting numbers of regions vector (1 x nr) of integer

Example

```
>> fg=gm_example_Sprinkler();
>> rg = gm_rg_CVM( fg)
rg =
  1×1 cell array
    {1×1 struct}
>> rg{1}
ans =
  struct with fields:
    Vr: [6×4 logical]
    Fr: [6×4 logical]
    Gr: [6×6 logical]
    cr: [1 1 1 -1 -1 -1]
>> rg{1}.Vr
ans =
  6×4 logical array
   1   1   0   0
   1   0   1   0
   0   1   1   1
   1   0   0   0
   0   1   0   0
   0   0   1   0
>> rg{1}.Fr
```

```
ans =
  6×4 logical array
   1   1   0   0
   1   0   1   0
   0   0   0   1
   1   0   0   0
   0   0   0   0
   0   0   0   0
>> rg{1}.Gr
ans =
  6×6 logical array
   0   0   0   1   1   0
   0   0   0   1   0   1
   0   0   0   0   1   1
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
```

The region graph created by the Cluster Variation Method(CVM) is the following:



2.4 Description of marginals

2.4.1 Definition

The marginal of a variable X_i is the probability distribution of this the variable, i.e. $\{p(x_i), \forall x_i\}$. It is also often called the belief of variable X_i, and denoted $b_i(x_i)$.
Similarly, we define the belief of a region R as the marginal probability distribution of the variables of that region, $b_R(x_{V_R})$. Since beliefs are probability distribution they satisfy the normalization property:

$$\sum_{x_{V_R}} b_R(x_{V_R}) = 1,$$

and they are consistent with the marginals computed on subsets of the variables X_{V_R} In particular, the belief of X_i can be obtained by marginalizing the belief of any region implying X_i:

$$b_i(x_i) = \sum_{x_{V_R \setminus \{i\}}} b_R(x_{V_R}), \forall R \; s.t. \; i \in V_R.$$

2.4.2 Matlab representation

The beliefs of all the variables of a factor graph `fg` are represented in a cell array called `b`.

`b`	cell array (1 x nv) b{i} is a vector (1 x fg.Card(i)) of double

Example

```
>> fg=gm_example_Sprinkler();
>> rg = gm_rg_CVM( fg);
>> b = gm_infer_GBP(fg, rg)
b =
```

```
  1×4 cell array
    {2×1 double}    {2×1 double}    {2×1 double}    {2×1 double}
>> fg.Name
ans =
  1×4 cell array
    {'Cloudy'}    {'Sprinkler'}    {'Rain'}    {'Wet'}
>> b{4}
ans =
    0.8015
    0.1985
```

Then the probability of variable 'Wet' (coded 4) to be true (coded 1) is estimated to 0.8015.

2.5 Description of beliefs on regions

2.5.1 Definition

We define the belief of a region R as the marginal probability distribution of the variables of that region, $b_R(x_{V_R})$.

2.5.2 Matlab representation

The beliefs of all the regions of a factor graph fg are represented in a cell array called br.
This cell array is structured as fg in sub factor graphs.

br	cell array (1 x n), n is the total number of regions in the factor graph br{i} is a cell array (1 x nr_i), nr_i is the number of regions in the sub factor graph i br{i}{j} is an array of double of size fg.Card(V_{R_j})), V_{R_j} is the set of variables in region R_j

Example

```
>> fg = gm_example_Sprinkler();
>> rg = gm_rg_CVM( fg);
>> [~, ~, br] = gm_infer_GBP(fg, rg)
br =
  1×1 cell array
    {1×6 cell}
>> fg.Name
ans =
  1×4 cell array
    {'Cloudy'}    {'Sprinkler'}    {'Rain'}    {'Wet'}
>> rg{1}.Vr
ans =
  6×4 logical array
   1   1   0   0
   1   0   1   0
   0   1   1   1
   1   0   0   0
   0   1   0   0
   0   0   1   0
>> % The region 1 contains 2 variables 'Cloudy' and 'Sprinkler'.
>> br{1}{1}
ans =
    0.2500    0.2500
    0.4500    0.0500
```

Looking at the beliefs of region 1 (containing 2 variables 'Cloudy' and 'Sprinkler'), the probability of 'Cloudy' set to yes (coded 1) and 'Sprinkler' set to on (coded 1) is estimated to 0.25.

2.6 Description of evidence

2.6.1 Definition

Evidence are observation (knowledge) of the state of some variables.

2.6.2 Matlab representation

The observations are represented in a vector called evidence.

evidence	observation value of variables, 0 for unknown vector (1 x nv) of integer

Example
In the Sprinkler example, assume that we know that variables Sprinkler and Rain are in state 1 (no sprinkler, no rain) and that the state of variables Cloudy and Wet.

```
>> evidence = [0 1 1 0]; % no sprinkler and no rain !
```

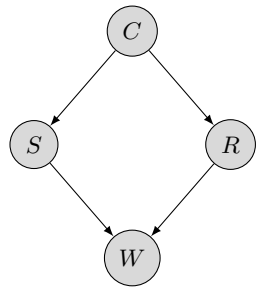
3 Factor graph generation functions

3.1 gm_example_Sprinkler

Description

A classical example of Bayesian network is the Sprinkler model. It models the relations between state of grass: wet/not (variable W), use of sprinkler: on/off (variable S), rain : yes/no (variable R) and cloudy sky : yes/no (variable C). The joint law is $p(C, S, R, W) = p(C)p(S \mid C)p(R \mid C)p(W \mid S, R)$.

p(C)	
C=true	0.5
C=false	0.5

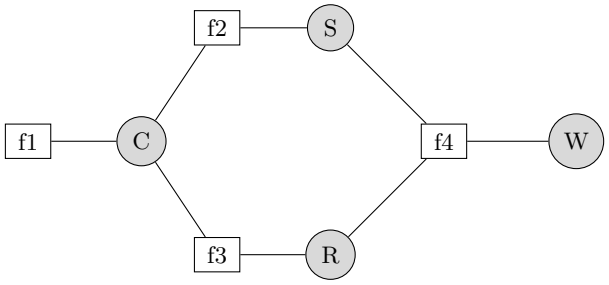


p(S|C)	S=true)	S=false
C=true	0.5	0.5
C=false	0.9	0.1

p(R|C)	R=true	R=false
C=true	0.8	0.2
C=false	0.2	0.8

p(W|S,R)	W=true	W=false
S=true, R=true	1	0
S=true, R=false	0.9	0.1
S=false, R=true	0.9	0.1
S=false, R=false	0.01	0.99

The factor graph of the sprinkler model has the following structure, with 4factors f1(C), f2(C,S), f3(C,R), f4(S,R,W).



To express the factor graph (Matlab structure fg) in the toolbox, we first define an ordering of the variables (C, S, R, W) that defines the coding of variables: 1 for C, 2 for S, 3 for R, 4 for W.
Then the cardinality of each variable is set in the Card attribut: fg.Card = [2 2 2 2]. Here all variable are boolean. We fix that 1 code false and 2 code true.
Then we express each factor.
Factor f1, only involves variable C: f1.V=1; f1.T=[0.5;0.5]; fg.F{1}=f1;

Factor f2, involves variables C,S : f2.V=[1,2]; f2.T=[0.5 0.5; 0.9 0.1]; fg.F{2}=f2;
Factor f3, involves variables C,S : f2.V=[1,3]; f2.T=[0.8 0.2; 0.2 0.8]; fg.F{3}=f3;
Factor f4, involves variables S,R,W : f2.V=[2,3,4]; f2.T(1,1,:)=[1 0];
f2.T(1,2,:)=[0.9 0.1]; f2.T(2,1,:)=[0.9 0.1]; f2.T(2,2,:)=[0.01 0.99]; fg.F{4}=f4;

Syntax

```
fg = gmdp_example_Sprinkler()
```

Evaluation

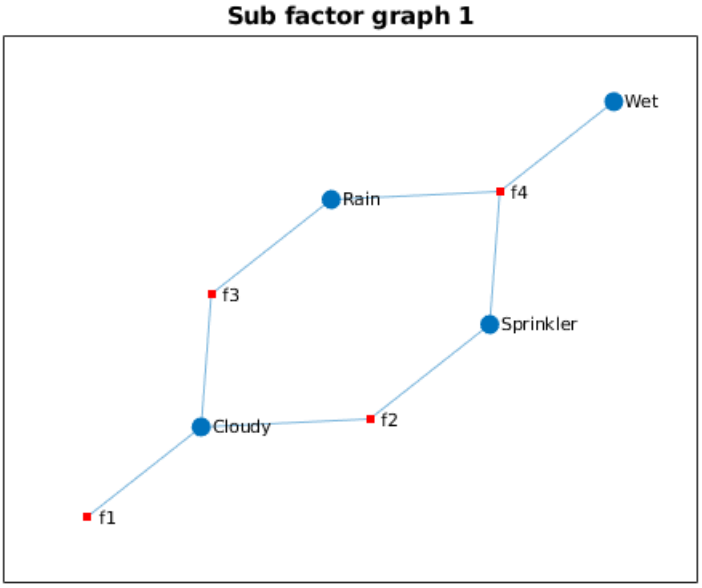
- **fg** : structure (see paragraph 2.2) defining the factor graph

Example

```
>> fg = gm_example_Sprinkler()
fg =
  struct with fields:
    Card: [2 2 2 2]
    Name: {'Cloudy'  'Sprinkler'  'Rain'  'Wet'}
     sfg: {[1×1 struct]}
>> sfg = fg.sfg{1}
sfg =
  struct with fields:
    Vcode: [1 2 3 4]
        F: {[1×1 struct]  [1×1 struct]  [1×1 struct]  [1×1 struct]}
       BG: [4×4 logical]
>> sfg.F{1}
ans =
  struct with fields:
    V: 1
    T: @()[0.5;0.5]
>> sfg.F{2}
ans =
  struct with fields:
    V: [1 2]
    T: @()[0.5,0.5;0.9,0.1]
>> sfg.F{3}
ans =
  struct with fields:
    V: [1 3]
    T: @()[0.8,0.2;0.2,0.8]
>> sfg.F{4}
ans =
  struct with fields:
    V: [2 3 4]
    T: @f4
>> sfg.F{4}.T()
ans(:,:,1) =
    1.0000    0.9000
    0.9000    0.0100
ans(:,:,2) =
         0    0.1000
    0.1000    0.9900
```

It is possible to visualize the structure of the unique sub factor graph associated to the Sprinkler model:

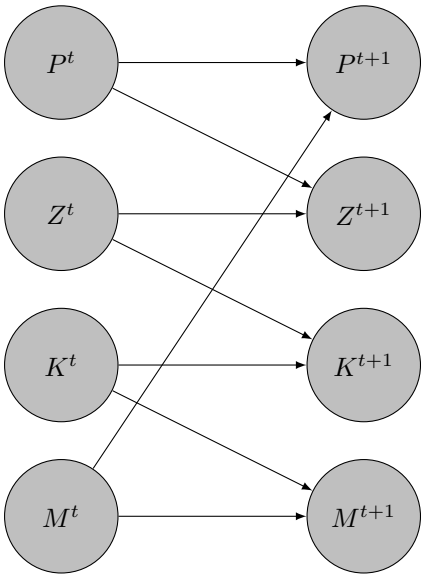
```
>> gm_plot_fg(fg)
```



3.2 gm_example_Ecology

Description

We consider a very simplified ecological network, 4 species: plankton (P), zooplankton (Z), krill (K), marine mammals (M) which population dynamic are interdependent. We modelize the annual population dynamic by the following Dynamic Bayesian Network (DBN).



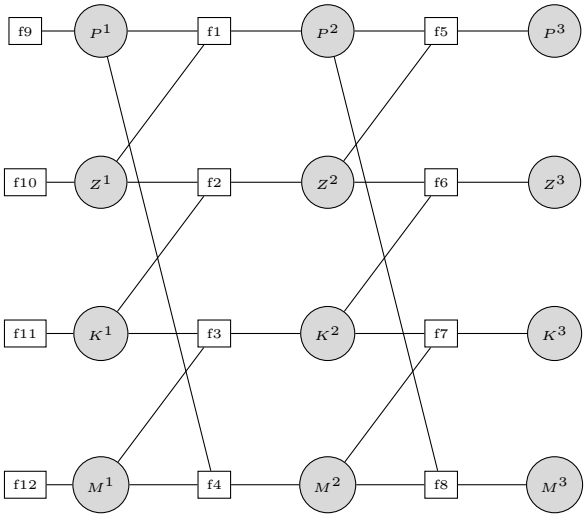
That is the population of plankton at time t+1 (P^{t+1}) is related to the population of plankton at time t (P^t) and the population of marine mammals (M^t) through feces, death... The other relations are prey-predator ones: marine mammals eat zooplankton that eat plankton.
The considering qualitative states are low (1), normal (2) and high (3) population.

The transition probabilities are simplified and are the same for all species. Considering the species V depending of also of species v, the transition $p(V^{t+1}|V^t, v^t)$ is defined by the following array T.

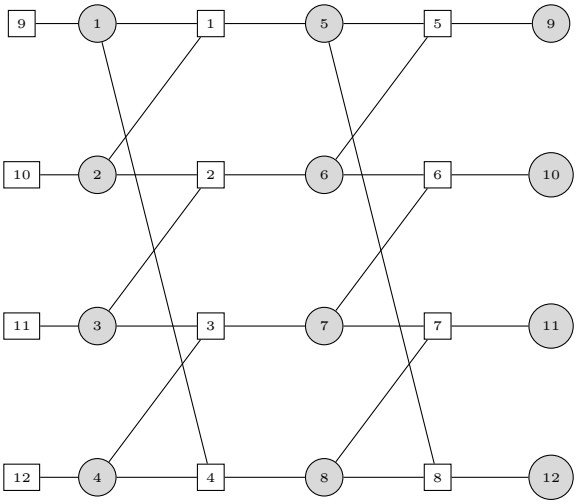
```
T(:,:,1) = [p1 p2 p4;   % $p(V^{t+1}=1|V^t, v^t)$
            p2 p3 p4;
            p4 p4 p4];
TT(:,:,2) =  [p4 p4 p4; % $p(V^{t+1}=2|V^t, v^t)$
              p2 p1 p2;
              p4 p4 p3];
T(:,:,3) = [p4 p4 p4;   % $p(V^{t+1}=3|V^t, v^t)$
            p4 p3 p2;
            p4 p2 p1];
```

For example purpose, we choose $p1 = 0.5, p2 = 0.2, p3 = 0.05, p4 = 0.01$.
Furthermore initial states are estimated quite normal $p(V^0 = 1) = 0.25, p(V^0 = 2) = 0.5, p(V^0 = 3) = 0.25$.

We consider 3 years (2 time steps).
Lets now represent the relations for the entire graphical model with factors graph and variables.



Representation of the problem
We define the list of variables as follows: $P^1, Z^1, K^1, M^1, P^2, Z^2, K^2, M^2, P^3, Z^3, K^3, M^3$.
Here is the same model with the coding numbers in Matlab for variables and factors.



Syntax

```
fg = gm_example_Ecology()
```

Arguments

No argument.

Evaluation

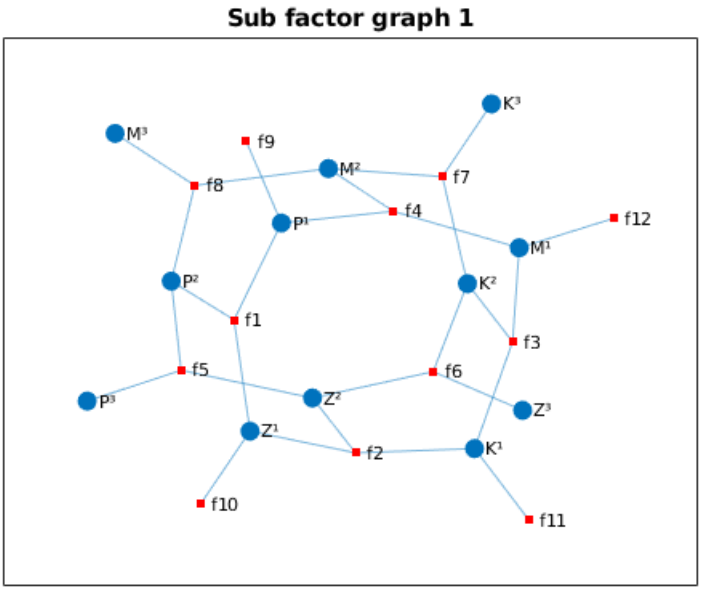
- **fg** : structure (see paragraph 2.2) defining the factor graph

Example

```
>> fg = gm_example_Ecology()
fg =
  struct with fields:
    Name: {1×12 cell}
    Card: [3 3 3 3 3 3 3 3 3 3 3 3]
     sfg: {[1×1 struct]}
>> sfg = fg.sfg{1}
sfg =
   struct with fields:
   Vcode: [1 2 3 4 5 6 7 8 9 10 11 12]
        F: {1×12 cell}
       BG: [12×12 logical]
>> sfg.F{1}
ans =
  struct with fields:
   V: [1 2 5]
    T: @()setTransition
>> sfg.F{1}.T()
ans(:,:,1) =
    0.5000    0.2000    0.0100
    0.2000    0.0500    0.0100
    0.0100    0.0100    0.0100
ans(:,:,2) =
    0.0100    0.0100    0.0100
    0.2000    0.5000    0.2000
    0.0100    0.0100    0.0500
ans(:,:,3) =
    0.0100    0.0100    0.0100
    0.0100    0.0500    0.2000
    0.0100    0.2000    0.5000
```

Plot of the factor graph.

```
>> gm_plot_fg(fg)
```



3.3 gm_example_CHMM

Description

A Hidden Markov Model (HMM) is composed of two variables: a sequence of hidden variables, which are never observed, and a sequence of observed variables (one per hidden variable) which state are known. The sequence of hidden variable is a Markov chain. HMM have been extended to Coupled HMM, where the hidden variables are multidimensional and whose dependencies can be represented as in a DBN.

The Coupled HMM example used in the toolbox model the dynamics of a pest that can spread on a landscape composed of N crop fields organized in a regular grid. The neighborhood of field i, denoted N_i, is the set of the 4 closest fields (or 3 or 2, on the borders and corners of the grid). $H_t^i \in \{0, 1\}$ $(1 \leq i \leq N,\ 0 \leq t \leq T)$ is the state of crop field i at time t. State 0 (resp. 1) represents the absence (resp. presence) of the pest in the field. Variable H_t^i depends on H_{t-1}^i and of the H_{t-1}^j such that j is in N_i. The conditional probabilities of survival and apparition of the pest in field i are parameterized by 3 parameters:

- ϵ, the probability of contamination from outside the landscape (long-distance dispersal).
- ρ, the probability that the pest spreads from an infected field $j \in N_i$ $(H_t^j = 1)$ to field i between time t and $t + 1$.
- ν, the probability that an infected field at time t remains infected at $t + 1$

We assume that contamination events from every neighbor fields are independent. Then, if I_t^i is the number of infected neighbors of field i at time t $(I_t^i = \sum_{j \in N_i} H_t^j)$, we have

$$P(X_{t+1}^i = 1 \mid X_t^i = 0, X_t^j, j \in N_i) = \epsilon + (1 - \epsilon)(1 - (1 - \rho)^{I_t^i})$$

and

$$P(X_{t+1}^i = 1 \mid X_t^i = 1, X_t^j, j \in N_i) = \nu + (1 - \nu)\left(\epsilon + (1 - \epsilon)(1 - (1 - \rho)^{I_t^i})\right).$$

The H_t^i are hidden variables. During monitoring, a binary variable O_t^n is observed: it takes value 1 if the pest has been observed and 0 otherwise. But error of detection is possible: we can have false negative observations (the pest is there but difficult to see so it was missed) or false positive observations (the pest was mixed up with another one). We define $P(O_t^i = 1 \mid X_t^i = 0) = f_p$ and $P(O_t^i = 0 \mid X_t^i = 1) = f_n$.

"Expert" values for the dynamics parameters in the case of a weed species can be $\epsilon = 0.15, \rho = 0.2, \nu = 0.5$. For the observations distributions, we take $f_p = 0.05$ and $f_n = 0.1$.

Syntax

```
fm = gmdp_example_CHMM()
fm = gmdp_example_CHMM(N)
fm = gmdp_example_CHMM(N, T)
```

Arguments

- **N** : number of fields in the square grid (default value: 9)
- **T** : number of time periods (default value: 3)

Evaluation

- **fg** : structure (see paragraph 2.2) defining the factor graph

Example

3.4 gm_example_DBN

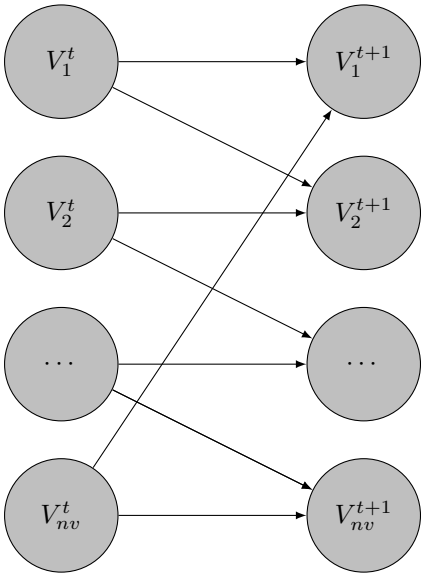
Description

A Dynamic Bayesian Network (DBN) is a particular Bayesian network where the same set of variables evolve through (discrete) time and the state of a variable in a given time period depends only on variables in the previous time period and the current one.

The function allows to create a simple Dynamic Bayesian Network with the following transition structure:

- no relation between variables into a time period,
- a variable V_i^{t+1} has two parents: V_i^t and V_{i-1}^t, expected V_1^{t+1} which as a unique parent V_1^t.

The following graph shows the relations between the variables of two consecutive time steps.



Syntax

```
global NS; fg = gm_example_DBN(nv, t)
```

Arguments

- **NS** : global variable, cardinality of each variable
- **nv** : number of variables in a time period
- **t** : number of time steps

Evaluation

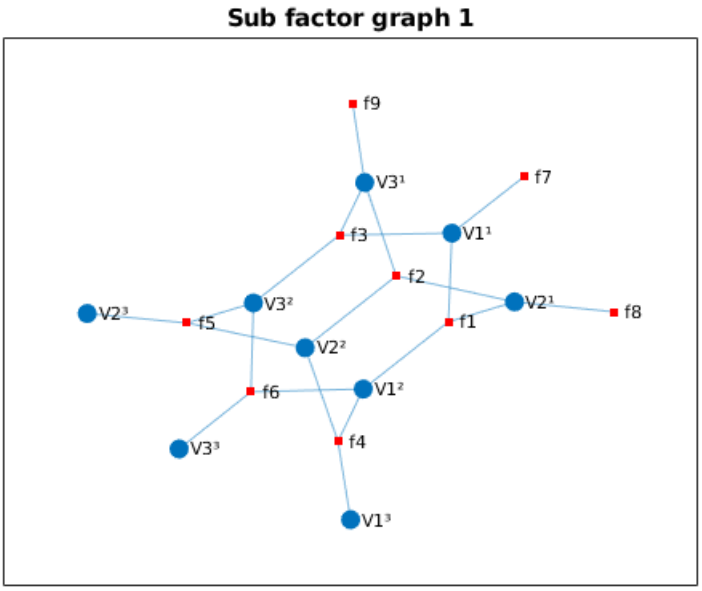
- **fg** : structure (see paragraph 2.2) defining the factor graph

Example

```
>> global NS; NS=2; fg = gm_example_DBN(3,2)
fg =
  struct with fields:
    Card: [2 2 2 2 2 2 2 2 2]
     sfg: {[1×1 struct]}
>> sfg = fg.sfg{1}
sfg =
  struct with fields:
    Vcode: [1 2 3 4 5 6 7 8 9]
        F: {1×9 cell}
       BG: [9×9 logical]
>> sfg.F{1}
ans =
  struct with fields:
    V: [1 2 4]
    T: [2×2×2 double]
```

Plot of the unique sub factor graph corresponding to the DBN, giving names to variables.

```
>> fg.Name = {'V1[1]','V2[1]','V3[1]','V1[2]','V2[2]','V3[2]','V1[3]','V2[3]','V3[3]'};
>> gm_plot_fg(fg)
```



3.5 gm_example_CHMM

Description

A Hidden Markov Model (HMM) is composed of two variables: a sequence of hidden variables, which are never observed, and a sequence of observed variables (one per hidden variable) which state are known. The sequence of hidden variable is a Markov chain. HMM have been extended to Coupled HMM, where the hidden variables are multidimensional and whose dependencies can be represented as in a DBN.

The Coupled HMM example used in the toolbox model the dynamics of a pest that can spread on a landscape composed of N crop fields organized in a regular grid. The neighborhood of field i, denoted N_i, is the set of the 4 closest fields (or 3 or 2, on the borders and corners of the grid). $H_t^i \in \{0, 1\}$ $(1 \leq i \leq N,\ 0 \leq t \leq T)$ is the state of crop field i at time t. State 0 (resp. 1) represents the absence (resp. presence) of the pest in the field. Variable H_t^i depends on H_{t-1}^i and of the H_{t-1}^j such that j is in N_i. The conditional probabilities of survival and apparition of the pest in field i are parameterized by 3 parameters:

- ϵ, the probability of contamination from outside the landscape (long-distance dispersal).
- ρ, the probability that the pest spreads from an infected field $j \in N_i$ $(H_t^j = 1)$ to field i between time t and $t + 1$.
- ν, the probability that an infected field at time t remains infected at $t + 1$

We assume that contamination events from every neighbor fields are independent. Then, if I_t^i is the number of infected neighbors of field i at time t $(I_t^i = \sum_{j \in N_i} H_t^j)$, we have

$$P(X_{t+1}^i = 1 \mid X_t^i = 0, X_t^j, j \in N_i) = \epsilon + (1 - \epsilon)(1 - (1 - \rho)^{I_t^i})$$

and

$$P(X_{t+1}^i = 1 \mid X_t^i = 1, X_t^j, j \in N_i) = \nu + (1 - \nu)\left(\epsilon + (1 - \epsilon)(1 - (1 - \rho)^{I_t^i})\right).$$

The H_t^i are hidden variables. During monitoring, a binary variable O_t^n is observed: it takes value 1 if the pest has been observed and 0 otherwise. But error of detection is possible: we can have false negative observations (the pest is there but difficult to see so it was missed) or false positive observations (the pest was mixed up with another one). We define $P(O_t^i = 1 \mid X_t^i = 0) = f_p$ and $P(O_t^i = 0 \mid X_t^i = 1) = f_n$.

"Expert" values for the dynamics parameters in the case of a weed species can be $\epsilon = 0.15, \rho = 0.2, \nu = 0.5$. For the observations distributions, we take $f_p = 0.05$ and $f_n = 0.1$.

Syntax

```
fm = gmdp_example_CHMM()
fm = gmdp_example_CHMM(N)
fm = gmdp_example_CHMM(N, T)
```

Arguments

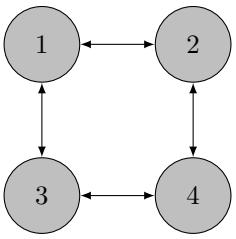
- **N** : number of fields in the square grid (default value: 9)
- **T** : number of time periods (default value: 3)

Evaluation

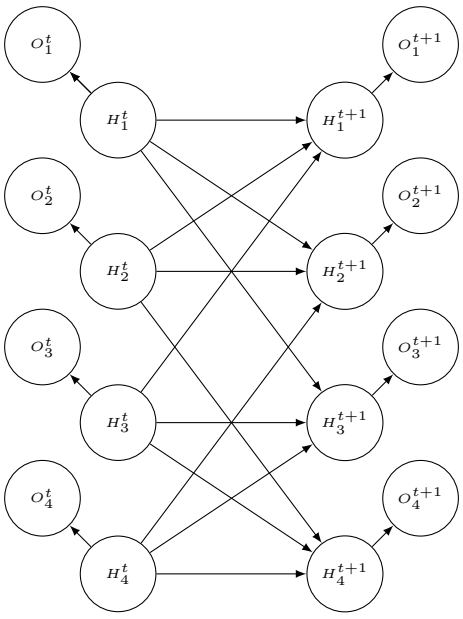
- **fg** : structure (see paragraph 2.2) defining the factor graph

Example

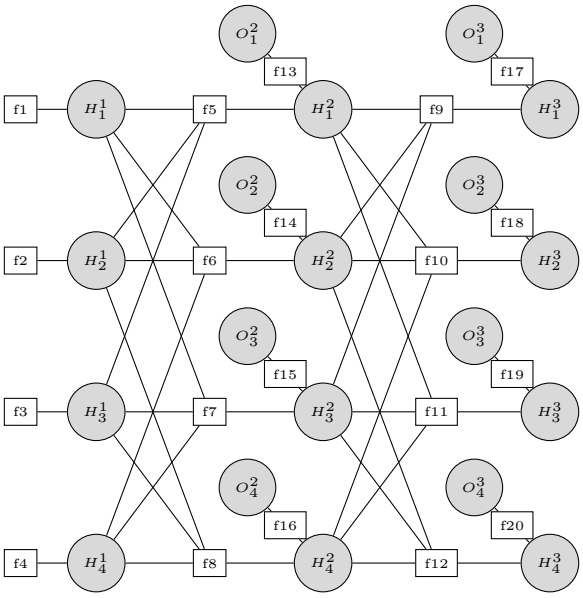
We consider a square grid of N=4 fields evolving during T=2 periods.
Here is the set of fields and the possible propagation of pest between them.



Let us now represent state and observation variables dependencies between 2 time steps.



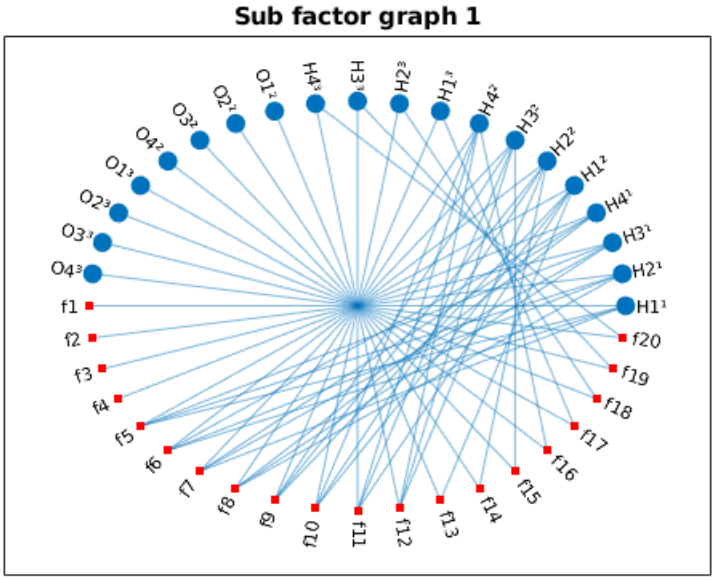
Here is the corresponding factor graph including the factors on the initial time step (f1 to f4) and two transitions.




```
>> fg = gm_example_CHMM(4,2)
fg =
  struct with fields:
    Card: [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
     sfg: {[1×1 struct]}
>> sfg = fg.sfg{1}
sfg =
  struct with fields:
       F: {1×20 cell}
    Vcode: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]
       BG: [20×20 logical]
>>  sfg.F{1}
ans =
  struct with fields:
    V: 1
    T: [2×1 double]
```

Plot of the unique sub factor graph, giving names to variables.

```
>> fg.Name = {'H1^1','H2^1','H3^1','H4^1', ...
              'H1^2','H2^2','H3^2','H4^2', ...
              'H1^3','H2^3','H3^3','H4^3', ...
              'O1^2','O2^2','O3^2','O4^2', ...
              'O1^3','O2^3','O3^3','O4^3'};
>> gm_plot_fg(fg, 1, 'circle')
```



3.6 gm_create_fg

Description

Create a factor graph structure, giving cardinalities of variables and a set of factors.
A factor graph is first created with 1 sub factor graph containing all factors. If this factor graph is not valid, an error is returned. If it is valid, and disconnected, the sub factor graph is separated. So this function may return a factor graph with several sub factor graphs.

Syntax

```
fg = gm_create_fg( Card, F, Name)
```

Arguments

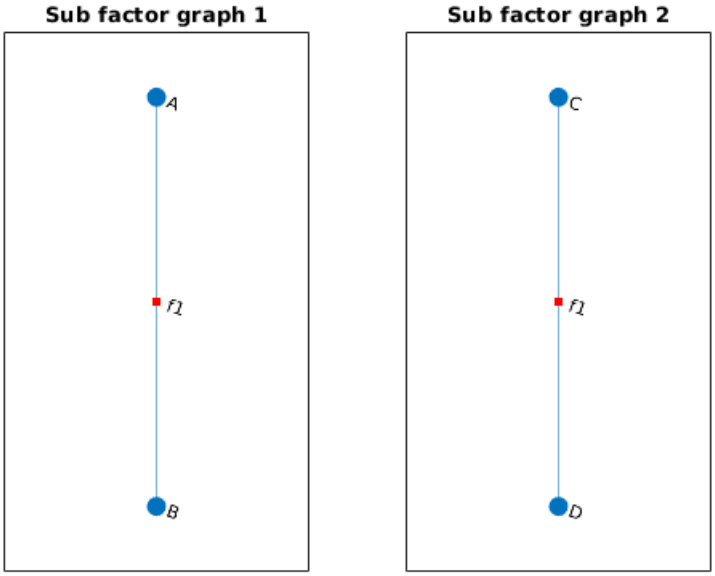
- **Card** : cardinality of variables, vector (1 x nv)of integer higher than 1
- **F** : set of factors, cell array of factor structure
- **Name** : set of variable names, cell array of char array (optional)

Evaluation

- **fg** : structure (see paragraph 2.2) defining the factor graph

Example

```
% Creation of a factor graph with a single sub factor graph,
% which is composed of 2 connex components
>> Card = [2 2 2 2]; % 4 variables
>> F1.V = [1 2];
>> F1.T = 0.5*ones(2);
>> F2.V = [3 4];
>> F2.T = 0.5*ones(2);
>> F = {F1, F2};
>> fg = gm_create_fg( Card, F, {'A','B','C','D'})
fg =
  struct with fields:
    Card: [2 2 2 2]
    Name: {'A'  'B'  'C'  'D'}
     sfg: {[1×1 struct]  [1×1 struct]}
>> fg.sfg{1}
ans =
  struct with fields:
    Vcode: [1 2]
        F: {[1×1 struct]}
       BG: [1 1]

gm_plot_fg( fg )
```



Here is an example of not coherent arguments.

```
>> fg = gm_example_Sprinkler();
>> fg_pb = gm_create_fg([2 2], fg.sfg{1}.F, '')
ERROR: F is not valid
ERROR: sfg{1}.BG not correct
fg_pb =
     []
```

3.7 gm_separate_fg

Description

Separate a factor graph in disconnected sub factor graphs, with each sub factor graph having a single connex component.

Syntax

```
fg = gm_separate_fg( fg )
```

Argument

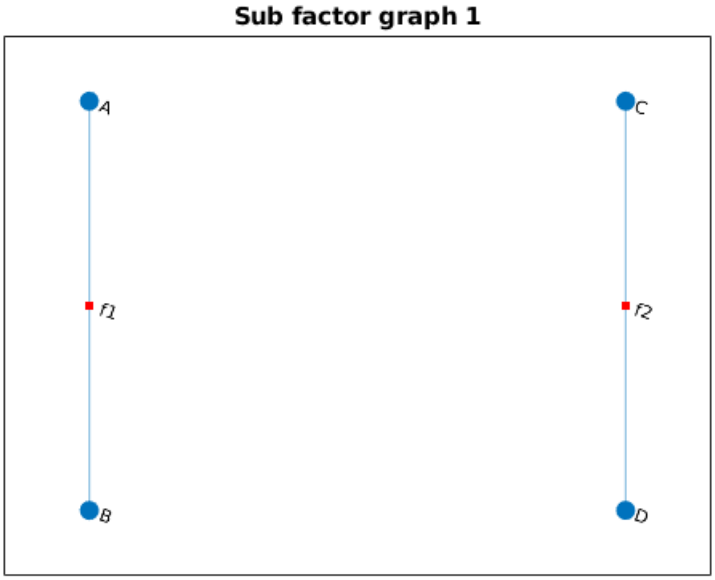
- **fg** : structure (see paragraph 2.2) defining a factor graph

Evaluation

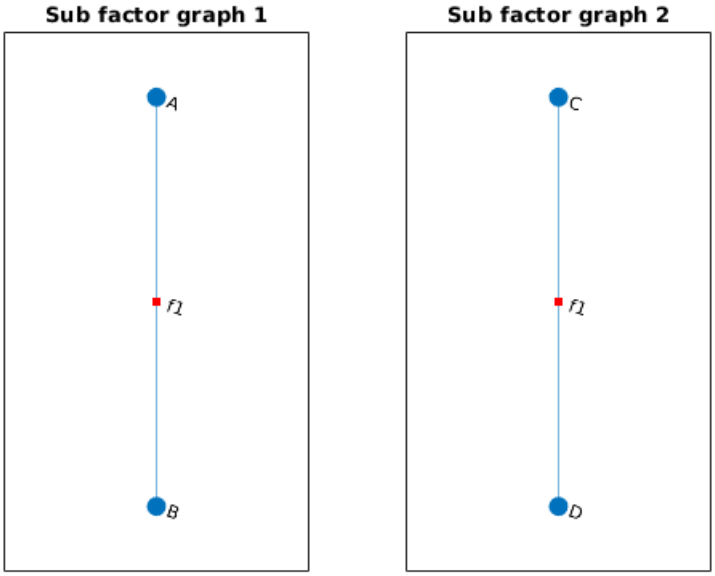
- **fg** : structure (see paragraph 2.2) defining a factor graph

Example

```
% Create by hand a simple disconnected factor graph
>> fg.Card = [2 2 2 2];
>> F1.V = [1 2];     % factor 1 on variable 1 and 2
>> F1.T = [ 0.5 0.5; 0.5 0.5];
>> F2.V  = [3 4];   % factor 2 on variable 3 and 4
>> F2.T = [ 0.5 0.5; 0.5 0.5];
>> sfg.Vcode = [1 2 3 4];
>> sfg.F = {F1, F2};
>> sfg.BG = setBG( length(sfg.Vcode), sfg.F);
>> fg.sfg{1} = sfg;

>> [is_OK, is_disconnected, msg] = gm_check_fg( fg )
WARNING:  sfg{1} is disconnected
is_OK =
  logical
   1
is_disconnected =
  logical
   1
msg =
  0×0 empty char array

>> gm_plot_fg(fg)
```




```
>> fg = gm_separate_fg( fg )
fg =
  struct with fields:
    Card: [2 2 2 2]
     sfg: {[1×1 struct]  [1×1 struct]}

>> gm_plot_fg(fg)
```



4 Generalized Belief Propagation

The toolbox implements a Generalized Belief Propagation (GBP) algorithm. There is a variety of algorithms, the Parent-to-Child algorithm is chosen. A main advantage of it is that the message-passing rules make no reference to region counting numbers, just as the standard Belief Propagation (BP) algorithm.

The GBP algorithm takes a factor graph *and a region graph* as inputs. Using the different functions we provide in the toolbox to compute a region graph for a given factor graph, one can either perform exact or approximate inference. If a *Junction Tree Region Graph* is computed, the GBP algorithm performs exact inference.
If, instead, a *Bethe Region Graph* is computed, the GBP algorithm becomes a standard *Belief Propagation* algorithm, which computes a Bethe approximation of the marginals on pairs of variables and on singletons.
Finally, if the *Cluster Variation Method Algorithm* is applied, flexibility is offered to the user, who can choose the 'large regions' of the region graph (those without parents), each choice leading to a different trade-off between accuracy and complexity.
These three region graph computation functions provide a gradient of approximation quality inversely proportional to their time/space complexity. Note that other Region Graph construction algorithms could be implemented in the toolbox, offering the possibility to develop and test new approximation approaches within the GBP algorithm.

Note also that the toolbox could be enriched by the addition of the *Child-to-Parent* and *Two-ways* algorithms described in [3].

4.1 gm_infer_GBP

Description

The parent-to-child algorithm generalizes the principles at the basis of the standard (exact) message passing algorithm on a tree structured factor graph. Messages are spread through regions instead of through factor nodes and variable nodes, and at convergence, beliefs on region are obtained as products of messages. More precisely the belief at any region R is the product of all the local factors in that region, multiplied by all the messages coming into region R from outside regions. There is one complication, however: to ensure that the algorithm is equivalent to minimizing the region graph free energy, we need to include additional messages into regions which are descendants of R from other parent regions that are not themselves descendants of region R.

There is just one kind of message, $m_{P\to R}(x_R)$, from a parent region P to a child region R. Each region R has a belief $b_R(x_R)$ given by:

$$b_R(x_R) = \prod_{a\in F_R} f_a(x_a).\Big(\prod_{P\in\mathcal{P}a(R)} m_{P\to R}(x_R)\Big).\Big(\prod_{D\in\mathcal{D}e(R)}\prod_{P'\in\mathcal{P}a(D)\setminus\mathcal{E}(R)} m_{P'\to D}(x_D)\Big)$$

where:

- $\mathcal{P}a(R)$ is the set of regions that are parents to region R,
- $\mathcal{D}e(R)$ is the set of all regions that are descendants of region R,

- $\mathcal{E}(R) = \{R\} \cup \mathcal{D}e(R)$ is the set of all regions that are descendants of R and also region R itself,
- $\mathcal{P}a(D)\backslash\mathcal{E}(R)$ is the set of all regions that are parents of region D except for region R itself or those those regions that are also descendants of region R.

The message-update rule is:

$$m_{P\to R} = \frac{\sum_{x_{P\setminus R}}(\prod_{a\in F_{P\setminus R}} f_a(x_a)).\prod_{(I,J)\in\mathcal{N}(P,R)} m_{I\to J}(x_J)}{\prod_{(I,J)\in\mathcal{D}(P,R)} m_{I\to J}(x_J)}$$

where:

- $\mathcal{N}(P,R)$ is the set of all connected pairs of regions (I,J) such that $I \notin \mathcal{E}(P)$, $J \in \mathcal{E}(P)\backslash\mathcal{E}(R)$.
- $\mathcal{D}(P,R)$ is the set of all connected pairs of regions (I,J) such that $I \in \mathcal{D}e(P)\backslash\mathcal{E}(R)$, $J \in \mathcal{E}(R)$.

Nota : The definition of $\mathcal{D}(P,R)$, in [3], is wrong. Replace definition of I by: $I \in \mathcal{E}(P)$.

Note that in the original article, updated messages are used as soon as they are computed, in the message update rule. Indeed, our coding implements:

$$m^{updated}_{P\to R} = \frac{\sum_{x_{P\setminus R}}(\prod_{a\in F_{P\setminus R}} f_a(x_a)).\prod_{(I,J)\in\mathcal{N}(P,R)} m^{old}_{I\to J}(x_J)}{\prod_{(I,J)\in\mathcal{D}(P,R)} m^{updated}_{I\to J}(x_J)}$$

where $m^{updated}_{P\to R}$ is the updated version of the message, while $m^{old}_{P\to R}$ is the old version. But, for this update rule to be operational, whenever we want to update a message $m^{updated}_{P\to R}$, every messages $m^{updated}_{I\to J}$ with $(I,J) \in \mathcal{D}(P,R)$ must have already been computed. This constrains the order of messages computation: we start by updating messages to the leaves of the region graph and continu by going up until the root. Several orders satisfy this rule, the same order is set and used all along the algorithm.

Syntax

```
[b, stop_by, br] = gm_infer_GBP ( fg , rg )
[b, stop_by, br] = gm_infer_GBP ( fg , rg, Nmax )
[b, stop_by, br] = gm_infer_GBP ( fg , rg, Nmax, epsilon )
[b, stop_by, br] = gm_infer_GBP ( fg , rg, Nmax, epsilon, damp )
[b, stop_by, br] = gm_infer_GBP ( fg , rg, Nmax, epsilon, damp, is_verbose )
```

Arguments

- **fg** : structure (see paragraph 2.2) defining the factor graph
- **rg** : structure (see paragraph 2.3) defining the region graph
- **Nmax** : maximum number of iterations, integer in [1 1000] (optional, default value: 100)
- **epsilon** : stop iteration when the maximal difference between messages between 2 iterations is less than epsilon, positive double, (optional, default 0.0001)
- **damp** : damp factor to update messages, damp = 0 means 'take new message, damp = 1 would mean 'take message from previous iteration', double in [0 1[(optional, default 0)
- **is_verbose** : if true, display additional information when running (optional, default false)

Evaluation

- **b** : cell array (1 x nv), beliefs of variables
- **stop_by** : string, stop criterion in {"Nmax", "epsilon"} (empty when fg.sfg if empty)
- **br** : cell array (1 x nr), beliefs of regions

Example

```
>> fg = gm_example_Sprinkler();
>> rg = gm_rg_CVM(fg);

>> [b, stop_by, br] = gm_infer_GBP ( fg , rg )
b =
  1×4 cell array
    {2×1 double}    {2×1 double}    {2×1 double}    {2×1 double}
stop_by =
  1×1 cell array
    {["epsilon"]}
br =
  1×1 cell array
    {1×6 cell}
>> b{4}(1) % probability of grass (variable 'Wet' coded 4) to be wet (coded 1)
ans =
    0.8015
>> br{1}{3}(1,1,1)  % 3 variables in region 3: 'Sprinkler', 'Rain', 'Wet'
% probability of sprinkler to be on, rain to be yes and grass to be wet
ans =
    0.3500

% In verbose mode, number of iteration are given
>> [b, stop_by] = gm_infer_GBP ( fg, rg, 10, 0.0001, 0, true );
n=1    delta=0.57143
n=2    delta=0
```

Coding

The GBP algorithm can be expressed using two functions :

- Prod : computes the product of the tables of the functions F_i, taking into account the variables V_i of of each function F_i.

  $$[h, v] = Prod(H, V)$$

  with $H = \{F_i\}$, list of function F_i's tables,
  $W = \{V_i\}$, list of indices of the variables of functions F_i.
  $h = \prod_i F_i$, the product of functions F_i,
  $V = \cup_i V_i$, the indices of the variables of h.

- Marg : computes the marginals of a function f defined on the set v_f of variables, by eliminating the set of variables v_e.

  $$[h, v] = Marg(f, v_f, v_e)$$

  with $v = v_f \backslash v_e$.

An other function, extend_message, is used in the gm_infer_GBP function. The function, if necessary, extends the result h of Prod function such that the scope of h is equal to the set of variables of region receiving the message computed using h.

Algorithm 1: Generalized Belief Propagation (GBP) algorithm

Input: fg : a factor graph
rg : a region graph
$Nmax$: maximum number of iterations (optional, default 100)
ϵ : stop iteration when all the differences between messages of two successive iterations are less than ϵ (optional, default 0.0001)
damp : damping factor $\in [0, 1[$ (optional, default 0)

Output: b : beliefs on variables (marginals)
stop_by : criterium that stopped message iteration
br : beliefs on regions

begin

$\delta \leftarrow +\infty$, $n \leftarrow 0$
Init $\mathcal{P}a$, $\mathcal{D}e$, $\mathcal{E}$, $\mathcal{N}$, $\mathcal{D}$ for all regions
Init *message_order* considering regions from leaves to root
Init messages $m^{updated}$ */* *with uniform distribution* */*
oscillation $\leftarrow 0$;

while $\delta > \epsilon$ *and* $n < Nmax$ **do**

$\delta \leftarrow 0$, $n \leftarrow n + 1$, $m^{old} = m^{updated}$
/* *Compute messages* */
foreach (P, R) *of message_order* **do**
/* *update messages* */
foreach $f_a \in F_{P \setminus R}$ **do**
$H \leftarrow H \cup \{f_a\}, W \leftarrow W \cup \{V_a\}$
foreach $(I, J) \in \mathcal{N}(P, R)$ **do**
$H \leftarrow H \cup \{m^{old}_{I \to J}\}, W \leftarrow W \cup \{V_J\}$
$[h, V] = Prod(H, W)$
$[h, V] = Marg(h, V, V \setminus V_R)$
$H \leftarrow h, W \leftarrow V$
foreach $(I, J) \in \mathcal{D}(P, R)$ **do**
$H \leftarrow H \cup \{1/m^{updated}_{I \to J}\}, W \leftarrow W \cup \{V_J\}$
/* $(1/m_{I \to J})(x_J) =_{def} 1/(m_{I \to J}(x_J))$ */
$m^{updated}_{P \to R} \leftarrow extend\_message(Prod(H, W))$
/* *Normalize messages with max and damping* */
foreach (P, R) *of message_order* **do**
$m^{updated}_{P \to R} \leftarrow normalize(m^{updated}_{P \to R})$
$m^{updated}_{P \to R} = (1 - damp).m^{updated}_{P \to R} + damp.m^{old}_{P \to R}$
/* *Compute stop condition* */
$\delta' \leftarrow \max_{(P,R) \in message\_order} \max_{x \in x_R} \mid m^{old}_{P \to R}(x) - m^{updated}_{P \to R}(x) \mid$
/* *Compute marginals* */
foreach *variable* $x_i \in \mathcal{X}$ **do**
$R \leftarrow$ a region with variable x_i and as few variables as possible
$b(x_i) = Marg(br(R), R, R \setminus \{i\})$

Normalization with maximum

A message can be seen as a table. We normalize each message by dividing by the maximum of the message's table. Then the maximum value is 1 in this table.

When iterating, dividing by a very small value may give NaN. To avoid this, values less than 0.0001, are set to 0.0001. This is not a problem with a normalization by maximum but it would be a problem with a normalization by the sum.

We evaluated that execution time are near for normalization by maximization and by sum (time for sum is just a little lower).

Considering the estimation of belief in examples run for comparison with libDAI (see 6.1), with sum the difference is about 10^{-15} for sum instead of 0.0017 when maximization.

Detection of oscillation
We try to automatically detect oscillations with a compromize in costs CPU time and memory. Neither criterion was satisfying, so we do not anymore try to detect oscillations but provide a function to visualize belief evolution all along iterations (function gm_plot_belief_evolution).
We have 2 examples of oscillation: (i) CHMM generated with gm_example_CHMM(4,2) with observations at time step 2 (variables 13, 14, 15, 16). It is to note that oscillations are different if observed states are 1 or 2; (ii) problem called Figure13 in TEST directory.

Order of messages
We first consider the leaf regions in the region graph.
Then we iteratively consider parents regions. Be careful that only parent regions that have all children yet considered are inserted in the order list. In other words, the insertion in the list order of those regions has to be delayed until all successors are in the list order.
Then messages $m_{P\to R}$ are updated starting from messages arriving to leaves of the region graph and then going backward until the root. This ensure that when updating $m_{P\to R}$ all messages $m_{I\to J}$ such that $(I,J)\in\mathcal{D}(P,R)$ have already been updated.

Improve convergence
In [3], under equation (153), there is the following sentence: "In practice, it often helps convergence to only step the messages part-way to their newly computed values. This simple heuristic can eliminate 'over-shooting' problems..." That is m^{update} is replaced by $m^{update} \leftarrow damp.m^{old}+(1-damp).m^{update}$.

4.2 gm_rg_JT

Description

If gm_infer_GBP is applied with the region graph computed by the gm_rgJT function, then the inference algorithm is equivalent to the Junction Tree (JT) algorithm.

The region graph construction for the JT method is based on the classical *variable elimination algorithm* for graphical models. The resulting region graph is a bipartite graph containing *large regions* and *small regions*. The large regions are constructed using variable elimination with respect to an elimination ordering of variables, o, either given by the user or using the default one (variable 1, then 2, ... until n are eliminated): Whenever a variable $o(i)$ is eliminated, a new large region is created, containing variable $o(i)$, as well as every factor containing variable $o(i)$ and the variables of this factor and no variables already eliminated (so doing, each factor is included in exactly one region). Once done, we eliminate all regions which variables are all included in another region (and add the factors of the eliminated region to the one which is kept).
Then, in the JT region graph construction, the small regions are obtained by computing intersections between large regions and keeping only those that do not correspond to an already created region. However, in order to perform an exact inference, the region graph should be a tree. Therefore, a *Maximum Spanning Tree* (MST) is computed, for the graph which vertices are the large regions and weighted edges are added between every pairs of large regions sharing at least one variable. The weight of an edge is precisely the number of variables shared by the two regions. Once the MST has been computed, the bipartite region graph is obtained by replacing each edge $R_i - R_j$ of the MST by two oriented edges $R_i \to S_{ij}$ and $R_j \to S_{ij}$ with S_{ij} the intersection of R_i and R_j.

Syntax

```
rg = gm_rg_JT( fg )
rg = gm_rg_JT( fg, var_order )
```

Arguments

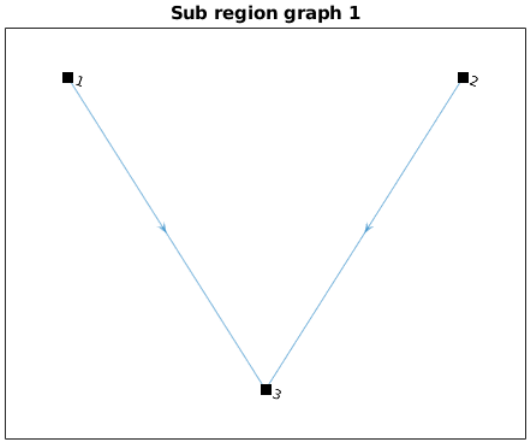
- **fg** : structure (see paragraph 2.2) defining the model
- **var_order** : a vector (1 x n), which is a reodering of the variables

Evaluation

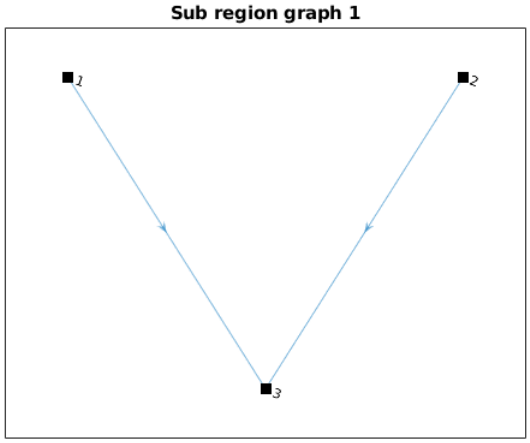
- **rg** : structure (see paragraph 2.3 for a description of the region graph)

Example

```
>> rg = gm_rg_JT( gm_example_Sprinkler() )
rg =
  1×1 cell array
    {1×1 struct}
>> rg{1}
ans =
  struct with fields:
    Vr: [3×4 logical]
    Fr: [3×4 logical]
    Gr: [3×3 logical]
    cr: [1 1 -1]
>> rg{1}.Vr
ans =
  3×4 logical array
   1   1   1   0
   0   1   1   1
   0   1   1   0
>> rg{1}.Fr
ans =
  3×4 logical array
   1   1   1   0
   0   0   0   1
   0   0   0   0
>> gm_plot_rg( rg)
```



```
>> rg = gm_rg_JT( gm_example_Sprinkler(), [4 3 2 1] )
rg =
  1×1 cell array
    {1×1 struct}
>> rg{1}
ans =
  struct with fields:
    Vr: [3×4 logical]
    Fr: [3×4 logical]
    Gr: [3×3 logical]
    cr: [1 1 -1]
>> rg{1}.Vr
ans =
  3×4 logical array
   0   1   1   1
   1   1   1   0
   0   1   1   0
>> rg{1}.Fr
ans =
  3×4 logical array
   0   0   0   1
   1   1   1   0
   0   0   0   0
>> gm_plot_rg( rg)
```



with this simple example, the region graph has also 3 regions with the same connections but regions are not similar: regions 1 and 2 exchanged.

Coding

Algorithm 2: GM_RG_JT Compute a region graph corresponding to a junction tree for a given variable ordering.

Input: fg : a factor graph
o : an ordering of the variables (optional)

Output: rg : a region graph with has the structure of a junction tree

begin

```
/* Use variable elimination to compute large regions                    */
```
rg.regions ← COMPUTE_LARGE_REGIONS(fg, o) ;
```
/* Use Kruskal's Maximum Spanning Tree algorithm to compute small regions
   */
```
rg ← COMPUTE_SMALL_REGIONS(rg) ;

Algorithm 3: COMPUTE_LARGE_REGIONS, Compute large regions (variables, factors) for JT, for a given elimination order (illustrated on a factor graph with only one sub factor graph)

Input: fg: a factor graph
o: an ordering of the variables

Output: LR: a set of regions, a region is a couple (S, F) with S (resp. (F) a set of variables (resp. factors)

begin
  for $k = 1 \dots n$ **do**
    /* F_k is the set of factors which scope contains $x_{o(k)}$ and no variable already eliminated */
    $F_k \leftarrow \{f \in \text{fg.sgf1.F}, x_{o(k)} \in Scope(f)\}$;
    $F_k \leftarrow F_k \cap \overline{(\cup_{j<k} F_j)}$;
    /* S_k contains the variables not yet eliminated and contained together with x_k in the scope of a factor already encountered */
    $S_k \leftarrow \left\{j \in 1, \dots, n, \exists f \in F_1 \cup \dots \cup F_k, (x_{o(k)}, x_j) \subseteq Scope(f)\right\}$;
    $S_k \leftarrow S_k \cap \overline{(o(1), \dots, o(k-1))}$;
  /* The large regions are built from the maximal elements of $\{S_k\}_k$ */
  for $k = 1 \dots n$ **do**
    if $\exists S_j$ *s.t.* $S_k \subseteq S_j$ **then**
      $F_j \leftarrow F_j \cup F_k$;
      $S_k \leftarrow \emptyset$, $F_k \leftarrow \emptyset$;
  $LR \leftarrow \{(S_k, F_k), (S_k, F_k) \neq (\emptyset, \emptyset)\}$

Algorithm 4: COMPUTE_SMALL_REGIONS: Compute small regions from large regions for JT and build the JT region graph (illustrated on a factor graph with only one sub factor graph)

/* Compute small regions corresponding to intersections between large regions and compute the region graph edges linking two large regions nodes to the small small region containing the intersection nodes */
Input: LR: a set of regions, a region is a couple (S, F) with S (resp. (F) a set of variables (resp. factors)

Output: rg: a region graph

/* First step: building of a weighted clique over the regions in LR */
$V = \{1, \dots, K\}$; /* |LR| = K */
/* The weights of the edges are the cardinal of the regions intersections */
$w(v_k, v_j) = |LR(j) \cap LR(k)|, \forall 1 \leq j, k \leq K$;
/* Second step: compute a maximum spanning tree of (V, w) */
$\mathcal{T} \leftarrow$MAXIMUM_SPANNING_TREE(V, w) ; /* Uses classical Kruskall's algorithm */
/* Third step: a small region of the JT region graph is created for each edge of maximum spanning tree of (V, w) */
for $e = (j, k) \in \mathcal{T}$ **do**
  $SR(e) \leftarrow$ LR$(j) \cap$ LR(k)
Regions = LR $\cup$ SR ;
rg.srg{1}.Vr = logical array, cell(i, j) true if variable j is in region i;
rg.srg{1}.F = logical array, cell(i, j) true if factor j is in region i;
rg.sfg{1}.cr = count numbers associate to regions;

4.3 gm_rg_BETHE

Description

The region graph corresponding to the Bethe approximation is a simple graph with two levels of regions, respectively *large regions* and *small regions*:

- There is one large region r_f per factor node f of the factor graph. r_f contains factor f as well as every variables i which are in the scope of f.
- There is one small region r_i per variable i of the factor graph.

Then, we build one directed edge from every large region nodes to the small region corresponding to the variables of the unique factor contained in the large region.

Syntax

```
rg = gm_rg_BETHE( fg )
```

Arguments

- **fg** : structure (see paragraph 2.2) defining the model

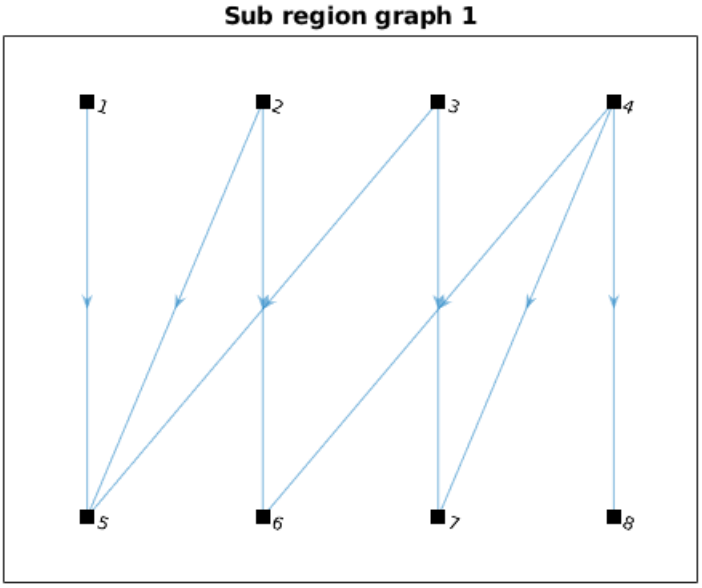
Evaluation

- **rg** : structure (see paragraph 2.3 for a description of the region graph)

Example

```
>> rg = gm_rg_BETHE( gm_example_Sprinkler() )
rg =
  1×1 cell array
    {1×1 struct}
>> rg{1}
ans =
  struct with fields:
    Vr: [8×4 logical]
    Fr: [8×4 logical]
    Gr: [8×8 logical]
    cr: [1 1 1 1 -2 -1 -1 0]
>> rg{1}.Vr
ans =
  8×4 logical array
   1   0   0   0
   1   1   0   0
   1   0   1   0
   0   1   1   1
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

```
>> rg{1}.Fr
ans =
  8×4 logical array
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
   0   0   0   0
   0   0   0   0
   0   0   0   0
   0   0   0   0
>> gm_plot_rg( rg )
```



Coding

Algorithm 5: GM_RG_BETHE: Compute the BETHE region graph (illustration for a factor graph with a single sub factor graph)

Input: fg : a factor graph
Output: rg : the region graph corresponding to the Bethe method
begin
  `/* Each large region corresponds to a factor and its variables */`
  LR $\leftarrow \cup_{f \in \text{fg.sg} f\{1\}} \{f, Scope(f)\}$;
  `/* Each small region corresponds to a single variable */`
  SR $\leftarrow \cup_{f \in \text{fg}} Scope(f)$;
  rg.regions $\leftarrow$ LR $\cup$ SR ;
  `/* Now, compute the graph edges linking large to small regions */`
  for $f \in LR$ **do**
    for $g \in SR$ **do**
      rg.srg{1}.Gr(f, g) = true if variable in g is in the scope of factor in LR ;

4.4 gm_rg_CVM

Description

The function builds a region graph with the Cluster Variation Method (CVM). It begins with a set of large regions $\mathcal{R}_0$ such that:

- every factor node f and every variable node i of the considered factor graph is included in at least one region $R \in \mathcal{R}_0$,
- no region $R \in \mathcal{R}_0$ is a subregion of any other region in $\mathcal{R}_0$.

Then the set of regions $\mathcal{R}_1$ is constructed by forming all possible intersections between regions in $\mathcal{R}_0$, but discarding from $\mathcal{R}_1$ any intersection regions that are sub-regions of other intersection regions in $\mathcal{R}_1$.

If possible, the set of regions $\mathcal{R}_2$ is constructed the same way from the intersection between regions in $\mathcal{R}_0 \cup \mathcal{R}_1$, but discarding any sub-regions that already appeared in $\mathcal{R}_1$ or that are sub-regions of other intersection regions in $\mathcal{R}_2$.

The procedure continues as long as there are new intersection regions. Finally the CVM set of regions is $\mathcal{R}_0 \cup \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_K$.

To form a region graph, connections between regions are constructed in the following way. For regions in $\mathcal{R}_1$, connect them to regions of $\mathcal{R}_0$ that are super regions. For a region R in $\mathcal{R}_2$, connect it to all regions in $\mathcal{R}_0$ and $\mathcal{R}_1$ that are super-regions of R, except for those regions in $\mathcal{R}_0$ that do not need a direct connection, because they are super-regions of regions in $\mathcal{R}_1$ that are also super-regions of R. Similar rules are followed for regions in $\mathcal{R}_3$ and so on.

Syntax

```
[rg, kR] = gm_rg_CVM( fg )
[rg, kR] = gm_rg_CVM( fg, regions)
```

Arguments

- **fg** : structure (see paragraph 2.2) defining the model
- **regions** : a matrix (nr x $(nv + nf)$), specifying regions for fg (optional)

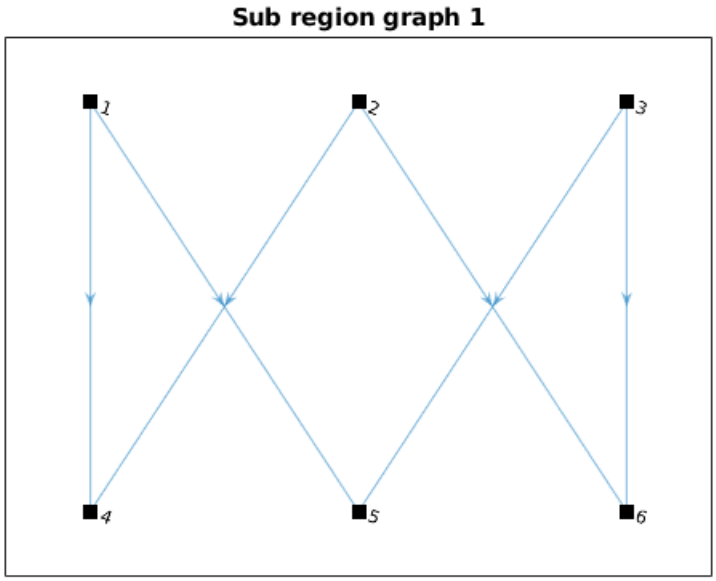
Evaluation

- **rg** : structure (see paragraph 2.3 for a description of the region graph)
- **kr** : vector (1 x nr), a level for each region

Example

```
>> fg = gm_example_Sprinkler();
>> rg = gm_rg_CVM( fg )
rg =
  1×1 cell array
    {1×1 struct}
>> rg{1}
ans =
  struct with fields:
    Vr: [6×4 logical]
    Fr: [6×4 logical]
    Gr: [6×6 logical]
    cr: [1 1 1 -1 -1 -1]
```

```
>> rg{1}.Vr
ans =
  6×4 logical array
   1   1   0   0
   1   0   1   0
   0   1   1   1
   1   0   0   0
   0   1   0   0
   0   0   1   0
>> rg{1}.Fr
ans =
  6×4 logical array
   1   1   0   0
   1   0   1   0
   0   0   0   1
   1   0   0   0
   0   0   0   0
   0   0   0   0
>> gm_plot_rg( rg )
```



To illustrate, specifying second argument regions, we require just 1 region with all variables and factors.

```
>> regions{1} = [1 1 1 1 1 1 1 1];
>> rg = gm_rg_CVM( fg , regions )
rg =
  1×1 cell array
    {1×1 struct}
>> rg{1}
ans =
  struct with fields:
    Vr: [1 1 1 1]
    Fr: [1 1 1 1]
    Gr: 0
    cr: 1
```

Coding

Algorithm 6: GM_RG_CVM Compute a region graph with Cluster Variation Method

Input: fg : a factor graph
regions : initial set of regions (level k=0) (optional)

Output: rg : a region graph
rk : ranks of regions in the region graph

begin

/ Check arguments validity and define an initial set of regions */*

if *regions provided* **then**

$\mathcal{R}_0 \leftarrow$ regions

else

$\mathcal{R}_0 \leftarrow$ extract regions from fg

(i) one region r_f per factor f

(ii) remove regions $r_{f'}$ whose variables are included in another region r_f and add factor f' of the deleted region to regions r_f

`/* Remove regions of` $\mathcal{R}_0$ `which are strict subregions of another region */`

$\mathcal{R}_0 \leftarrow Purge\_subregions(\mathcal{R}_0)$;

`/* Initialize Region graph` $G = (V, E)$ `*/`

$G \leftarrow (V = \mathcal{R}_0, E = \emptyset)$;

`/* Compute intersection regions */`

$\mathcal{R}' \leftarrow Intersections(\mathcal{R}_0)$;

$\mathcal{R}' \leftarrow Purge\_subregions(\mathcal{R}')$;

`/* Remove every regions` $R \in \mathcal{R}'$ `which belong to` $\mathcal{R}_0$ `*/`

$\mathcal{R}' \leftarrow Purge\_equal(\mathcal{R}', \mathcal{R}_0)$;

$k \leftarrow 0$;

`/* Main loop */`

while $\mathcal{R}' \neq \emptyset$ **do**

$k \leftarrow k + 1$;

`/* Update regions */`

$\mathcal{R}_k \leftarrow \mathcal{R}_{k-1} \cup \mathcal{R}'$;

`/* Update graph */`

$E \leftarrow E \cup \{(R, R') \in V \times \mathcal{R}', R' \subseteq R\}$;

$V \leftarrow V \cup \mathcal{R}'$;

`/* Update rk */`

$rk \leftarrow [kr, \underbrace{k, \ldots, k}_{|\mathcal{R}'|}]$;

`/* Next step: Compute intersection regions */`

$\mathcal{R}' \leftarrow Intersections(\mathcal{R}_k)$;

$\mathcal{R}' \leftarrow Purge\_subregions(\mathcal{R}')$;

$\mathcal{R}' \leftarrow Purge\_equal(\mathcal{R}', \mathcal{R}_k)$;

rg.Vr $\leftarrow \mathcal{R}_k \cap variables$;

rg.Fr $\leftarrow \mathcal{R}_k \cap factors$;

rg.Gr $\leftarrow (V, E)$;

4.5 gm_include_evidence

Description

The function allows to take into account the known states (observations) of some variables. Th ouput factor graph corresponds to the joint distribution of the variables conditionally to the evidence.

Syntax

```
fg = gm_include_evidence( fg, evidence)
```

Argument

- **fg** : structure (see paragraph 2.2) defining a factor graph
- **evidence** : observations, vector (1 x nv) of integer (see paragraph 2.6), 0 for unknown state

Evaluation

- **fg** : structure (see paragraph 2.2) defining a factor graph

Example

```
>> evidence = [1 0 0 0]; % Weather is cloudy
>> fg = gm_include_evidence( gm_example_Sprinkler(), evidence)
fg =
  struct with fields:
        Card: [2 2 2 2]
        Name: {'Cloudy'  'Sprinkler'  'Rain'  'Wet'}
         sfg: {[1×1 struct]}
    evidence: [1 0 0 0]

>> rg = gm_rg_JT( fg )
rg =
  1×1 cell array
    {1×1 struct}

>> [b, stop_by] = gm_infer_GBP ( fg , rg )
b =
  1×4 cell array
    {2×1 double}    {2×1 double}    {2×1 double}    {2×1 double}
stop_by =
  1×1 cell array
    {["epsilon"]}

>> b{4}(1) % Probability that grass is wet if weather is cloudy
ans =
    0.8510
```

Note that evidence can disconnect a sub factor graph in several sub factor graphs.

```
>> evidence = [ 0 1 1 0]; % Rain and Sprinkler !
>> fg = gm_include_evidence( gm_example_Sprinkler(), evidence)
fg =
  struct with fields:
        Card: [2 2 2 2]
        Name: {'Cloudy'  'Sprinkler'  'Rain'  'Wet'}
         sfg: {[1×1 struct]  [1×1 struct]}
    evidence: [0 1 1 0]

>> rg = gm_rg_JT( fg )
rg =
  1×2 cell array
    {1×1 struct}    {1×1 struct}

>> [b, stop_by] = gm_infer_GBP ( fg , rg )
b =
  1×4 cell array
    {2×1 double}    {2×1 double}    {2×1 double}    {2×1 double}
stop_by =
  1×2 cell array
    {["epsilon"]}    {["epsilon"]}

>> b{4}(1) % Probability that grass is wet if rain and sprinkler
ans =
     1
```

Coding

Algorithm 7: GM_INCMUDE_EVIDENCE: Include evidence in a factor graph

Input: fg : a factor graph
evidence : evidences, cell array 1xnv

Output: fg : a factor graph

```
begin
    Vobs ←set of variables with evidence
    Reduce factors
    foreach factor a ∈ F do
        if Va \ Vobs not empty then
            Va ← Va \ Vobs
            Ta ← reduction of Ta
        else
            a ← empty
    Remove empty factors
    Add observed factors
    foreach variable v ∈ Vobs do
        add new factor with V=v, T=0, except T(v)=evidence{v}
```

4.6 gm_plot_belief_evolution

Description

Plot the evolution of belief for a set of pair (variable, state) during a range of maximum number of iteration (argument Nmax of gm_infer_GBP) of GBP.

Syntax

```
bt = gm_plot_belief_evolution ( V, S, i_start, i_end, fg, rg, epsilon, damp )
```

Argument

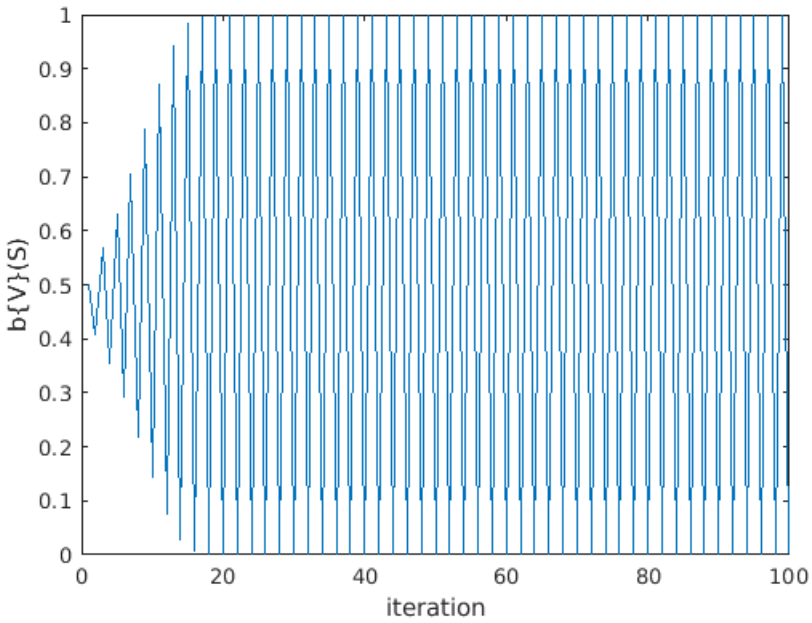
- **V** : vector of variable
- **S** : vector of state
- **i_start** : first Nmax considered
- **i_end** : last Nmax considered
- **fg** : structure (see paragraph 2.2) defining a factor graph
- **rg** : structure (see paragraph 2.3) defining a region graph
- **epsilon** : see epsilon argument of gm_infer_GBP
- **damp** : see damp argument of gm_infer_GBP

Evaluation

- **bt** : bt is a matrix such that bt(i, j) is p (V(j)=S(i)) at iteration i_start-1+j

Example

```
>> fg = gm_example_CHMM(4,2);
% observation at time step 2
>> evidence=zeros(1, length(fg.Card)); evidence(13:16)=2;
>> fge = gm_include_evidence( fg, evidence);
>> rge = gm_rg_CVM( fge);
>> bt = gm_plot_belief_evolution ( 1, 1, 1, 100, fge, rge, 0.0001, 0.0 );
```



5 Utilities

5.1 gm_check_fg

Description

Check that:

- variables appear in sub factor graphs
- sub factor graphs are disconnected
- each sub factor graph is valid

Syntax

```
[is_OK, msg] = gm_check_fg( fg )
```

Argument

- **fg** : structure (see paragraph 2.2) defining a factor graph

Evaluation

- **is_OK** : `true` if fg is valid, `false` otherwise
- **msg** : cell vector of error message(s)

Example

```
>> fg=gm_example_Sprinkler();
>> [is_OK, msg] = gm_check_fg( fg )
is_OK =
  logical
   1
msg =
     []
```

5.2 gm_check_rg

Description

Check that:

- each sub factor graph has an associated sub region graph,
- each sub region graph is valid (types of fields, each variable in a region, each factor in a region, sum of counting numbers equals to 1).

Syntax

```
[is_OK, msg] = gm_check_rg( rg, fg )
```

Argument

- **fg** : structure (see paragraph 2.2) defining a factor graph
- **rg** : structure (see paragraph 2.3) defining a region graph

Evaluation

- **is_OK** : `true` if rg is valid, `false` otherwise
- **msg** : cell vector of error message(s)

Example

```
>> rg = gm_rg_CVM( gm_example_Sprinkler() );
>> [is_OK, msg] = gm_check_rg( rg, fg )
is_OK =
  logical
   1
msg =
     []
```

5.3 gm_check_rg_JT

Description

Check if:

- rg is a junction graph (GBP acts as Loopy Belief Propagation algorithm),
- rg is a junction tree (GBP acts as Junction Tree algorithm).

For validity of rg, call gm_check_rg function.

Syntax

```
[is_JT, is_JG, msg] = gm_check_rg_JT( rg )
```

Argument

- **rg** : structure (see paragraph 2.3) defining a region graph

Evaluation

- **is_JT** : `true` if all sub region graphs are junction trees, `false` otherwise
- **is_JG** : `true` if all sub region graphs are junction graphs, `false` otherwise
- **msg** : cell vector of error message(s)

Example

```
>> fg = gm_example_Sprinkler();
>> rg = gm_rg_BETHE( fg)
rg =
  1×1 cell array
    {1×1 struct}
>> [is_JT, is_JG, msg] = gm_check_rg_JT( rg )
is_JT =
  logical
   0
is_JG =
  logical
   1
msg =
  0×0 empty char array>> [is_OK, msg] = gm_check_rg( rg, fg )
is_OK =
  logical
   1
msg =
     []

>> fge=gm_include_evidence(fg,[0 1 1 0]);
>> rge = gm_rg_BETHE( fge )
rge =
  1×2 cell array
    {1×1 struct}    {1×1 struct}
>> [is_JT, is_JG, msg] = gm_check_rg_JT( rge )
is_JT =
  logical
   1
is_JG =
  logical
   1
msg =
  0×0 empty char array
```

5.4 gm_plot_fg

Description

Plot sub factor graph(s). Variables are represented by large blue circles. Factors are represented by small red squares.
If variables have no Names they are called '1', '2' ... Factors of a sub factor graph are called 'f1', 'f2' ...

Syntax

```
gm_plot_fg( fg )
gm_plot_fg( fg , vsfg )
gm_plot_fg( fg , vsfg, layout )
```

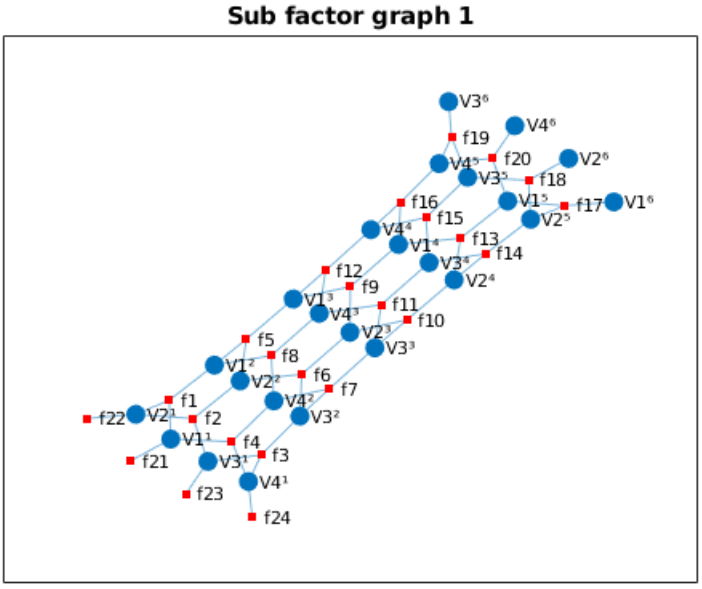
Argument

- **fg** : structure (see paragraph 2.2) defining a factor graph
- **vsfg** : set of sub factor graph, a vector of integer

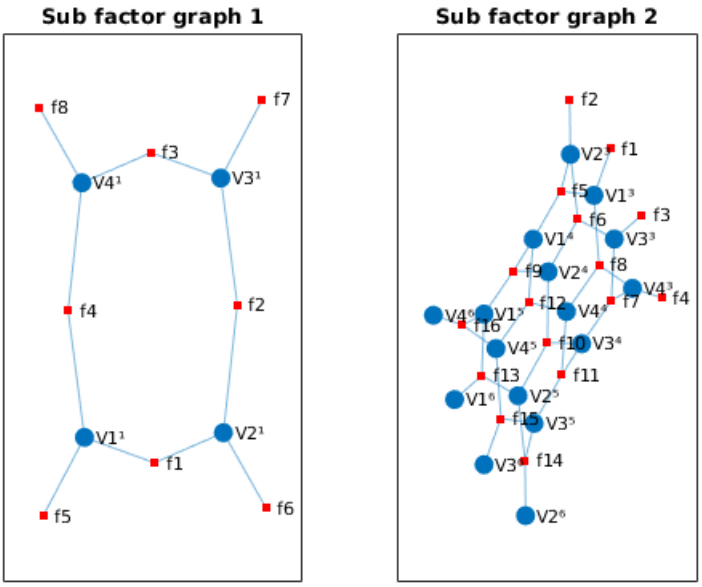
- **layout** : a layout [default 'auto'], possible value: 'auto', 'circle', 'force', 'layered', 'subspace', 'force3', 'subspace3'

Example

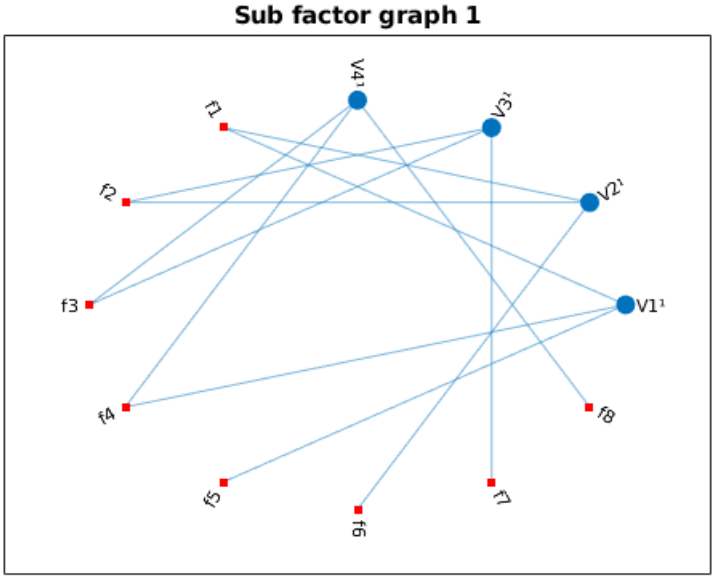
```
>> global NS; NS=3; fg = gm_example_DBN(4,5)
fg =
  struct with fields:
    Card: [3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
     sfg: {[1×1 struct]}
>> fg.Name = {'V1¹','V2¹','V3¹','V4¹', ...
              'V1²','V2²','V3²','V4²', ...
              'V1³','V2³','V3³','V4³', ...
              'V1⁴','V2⁴','V3⁴','V4⁴', ...
              'V1⁵','V2⁵','V3⁵','V4⁵', ...
              'V1⁶','V2⁶','V3⁶','V4⁶'}
>> gm_plot_fg(fg)
```




```
% States at period 2 are observed, the graph is cut in two
>> evidence=zeros(1,length(fg.Card)); evidence(5:8)=1;
>> fge=gm_include_evidence(fg, evidence)
fge =
  struct with fields:
        Card: [3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
         sfg: {[1×1 struct]  [1×1 struct]}
    evidence: [0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

>> gm_plot_fg(fge)
Evidence: 0  0  0  0  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```




```
% Improve visualisation of sub factor graph 1, changing layout
>> gm_plot_fg(fge, 1, 'circle')
Evidence: 0  0  0  0  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```



5.5 gm_plot_rg

Description

Plot sub region graph(s).
Regions are represented by large black squares.
Regions of a sub factor graph are called '1', '2' ...

Syntax

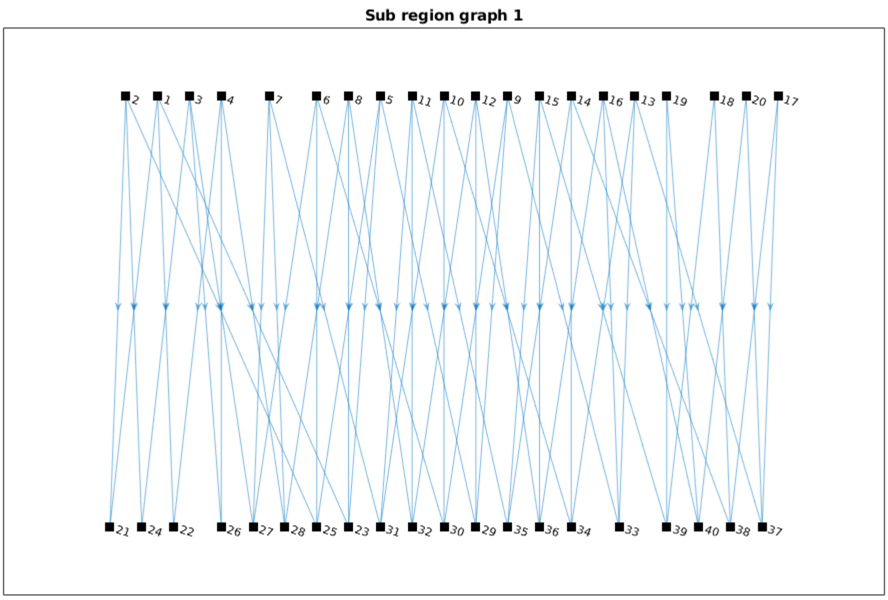
```
gm_plot_rg( fg )
gm_plot_rg( rg , vsfg )
gm_plot_rg( rg , vsfg, layout )
```

Argument

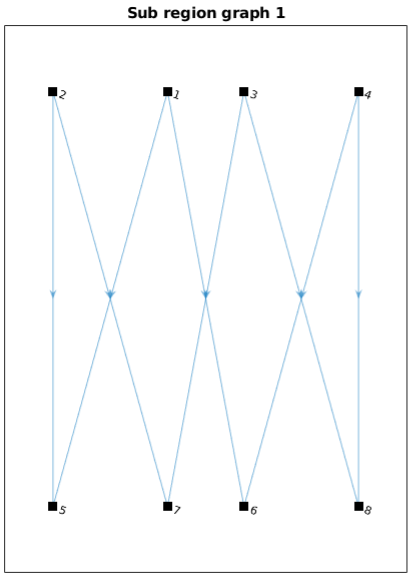
- **rg** : structure (see paragraph 2.3) defining a region graph
- **vsfg** : set of sub region graph, a vector of integer
- **layout** : a layout [default 'auto'], possible value: 'auto', 'circle', 'force', 'layered', 'subspace', 'force3', 'subspace3'

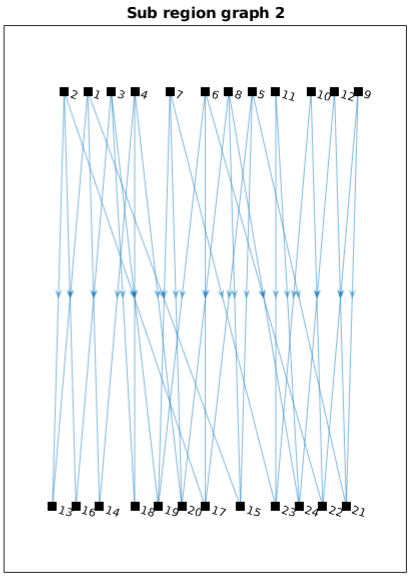
Example

```
>> global NS; NS=3; fg = gm_example_DBN(4,5)
fg =
  struct with fields:
    Card: [3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
     sfg: {[1×1 struct]}
>> rg = gm_rg_CVM( fg )
rg =
  1×1 cell array
    {1×1 struct}
>> gm_plot_rg(rg)
```



```
% States at period 2 are observed, the graph is cut in two
>> evidence = zeros(1,length(fg.Card)); evidence(5:8)=1;
>> fge = gm_include_evidence(fg, evidence)
fge =
  struct with fields:
        Card: [3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
         sfg: {[1×1 struct]  [1×1 struct]}
    evidence: [0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
>> rge = gm_rg_CVM( fge )
rge =
  1×2 cell array
    {1×1 struct}    {1×1 struct}
>> gm_plot_rg(rge)
```





5.6 gm_read_fg

Description

Read an UAI file containing a factor graph. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#model .

Syntax

```
fg = gm_read_rg( file_name )
```

Argument

- **file_name** : file name to read.

Evaluation

- **fg** : structure (see paragraph 2.2) defining a factor graph.

Example

```
>> gm_write_fg( gm_example_Sprinkler(), 'Sprinkler.uai')
>> fg = gm_read_fg( 'Sprinkler.uai' )
fg =
  struct with fields:
    Card: [2 2 2 2]
     sfg: {[1×1 struct]}
```

Here is the file 'Sprinkler.uai'.

```
MARKOV
4
2 2 2 2
4
1 0
2 0 1
2 0 2
3 1 2 3

2 0.5000 0.5000
4 0.5000 0.5000 0.9000 0.1000
4 0.8000 0.2000 0.2000 0.8000
8 1.0000 0.0000 0.9000 0.1000 0.9000 0.1000 0.0100 0.9900
```

5.7 gm_read_evidence

Description

Read an UAI file containing evidences. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#evidence .

Syntax

```
evidence = gm_read_evidence( file_name, nb_vars )
```

Argument

- **file_name** : file name to read.
- **nb_vars** : number of variables (just to check validity).

Evaluation

- **evidence** : observation of the state of some variables (see paragraph 2.6).

Example

```
>> gm_write_evidence( [0 1 0 1], 'Sprinkler.uai.evid')
>> evidence = gm_read_evidence(  'Sprinkler.uai.evid', 4)
evidence =
     0     1     0     1
```

Here is the file 'Sprinkler.uai.evid'.

```
1
2 1 1.0000 3 1.0000
```

5.8 gm_read_MAR

Description

Read an UAI file containing marginals. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#res .

Syntax

```
b = gm_read_MAR( file_name )
```

Argument

- **file_name** : file name to read.

Evaluation

- **b** : marginals (belief of variables) (see paragraph 2.4).

Example

```
>> for v=1:4; MAR{v}=[1; 0]; end
>> gm_write_MAR( MAR, 'Sprinkler.uai.MAR')
>> b = gm_read_MAR( 'Sprinkler.uai.MAR' )
b =
  1×4 cell array
    {2×1 double}    {2×1 double}    {2×1 double}    {2×1 double}
```

Here is the file 'Sprinkler.uai.MAR'.

```
MAR
1
4 2 1.0000 0.0000 2 1.0000 0.0000 2 1.0000 0.0000 2 1.0000 0.0000
```

5.9 gm_read_BEL

Description

Read an UAI file containing beliefs of regions. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#res .

Syntax

```
br = gm_read_BEL( file_name, fg, rg)
```

Argument

- **file_name** : file name to read.
- **fg** : structure (see paragraph 2.2) defining a factor graph.
- **rg** : structure (see paragraph 2.3) defining a region graph.

Evaluation

- **br** : beliefs of regions (see paragraph 2.5).

Example

```
>> fg = gm_example_Sprinkler();
>> rg = gm_rg_CVM(fg);
>> [~, ~, BEL] = gm_infer_GBP( fg, rg);
>> gm_write_BEL( BEL, 'Sprinkler.uai.BEL')
>> br = gm_read_BEL( 'Sprinkler.uai.BEL', fg, rg )
br =
  1×1 cell array
    {1×6 cell}
```

Here is the file 'Sprinkler.uai.BEL'.

```
BEL
1
6 4 0.2500 0.2500 0.4500 0.0500 4 0.4000 0.1000 0.1000 0.4000 8 0.3500 0.0000 0.3150 ...
   ... 0.0350 0.1350 0.0150 0.0015 0.1485 2 0.5000 0.5000 2 0.7000 0.3000 2 0.5000 0.5000
```

5.10 gm_write_fg

Description

Write an UAI file containing a factor graph. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#model .

Syntax

```
gm_write_rg( fg, file_name )
```

Argument

- **fg** : structure (see paragraph 2.2) defining a factor graph.
- **file_name** : file name to write.

Example

```
>> gm_write_fg( gm_example_Sprinkler(), 'Sprinkler.uai')
```

Here is the file 'Sprinkler.uai'.

```
MARKOV
4
2 2 2 2
4
1 0
2 0 1
2 0 2
3 1 2 3

2 0.5000 0.5000
4 0.5000 0.5000 0.9000 0.1000
4 0.8000 0.2000 0.2000 0.8000
8 1.0000 0.0000 0.9000 0.1000 0.9000 0.1000 0.0100 0.9900
```

5.11 gm_write_evidence

Description

Write an UAI file containing evidences. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#evidence .

Syntax

```
gm_read_evidence( evidence, file_name )
```

Argument

- **evidence** : observation of the state of some variables (see paragraph 2.6).
- **file_name** : file name to write.

Example

```
>> gm_write_evidence( [0 1 0 1], 'Sprinkler.uai.evid')
```

Here is the file 'Sprinkler.uai.evid'.

```
1
2 1 1.0000 3 1.0000
```

5.12 gm_write_MAR

Description

Write an UAI file containing marginals. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#res .

Syntax

```
gm_write_MAR( b, file_name )
```

Argument

- **b** : marginals (belief of variables) (see paragraph 2.4).
- **file_name** : file name to write.

Example

```
>> for v=1:4; MAR{v}=[1; 0]; end
>> gm_write_MAR( MAR, 'Sprinkler.uai.MAR')
```

Here is the file 'Sprinkler.uai.MAR'.

```
MAR
1
4 2 1.0000 0.0000 2 1.0000 0.0000 2 1.0000 0.0000 2 1.0000 0.0000
```

5.13 gm_write_BEL

Description

Write an UAI file containing beliefs of regions. For a complete UAI format description, see: http://www.cs.huji.ac.il/project/PASCAL/fileFormat.php#res .

Syntax

```
gm_write_BEL( br, file_name)
```

Argument

- **br** : beliefs of regions (see paragraph 2.5).
- **file_name** : file name to write.

Example

```
>> fg = gm_example_Sprinkler();
>> rg = gm_rg_CVM(fg);
>> [~, ~, BEL] = gm_infer_GBP( fg, rg);
>> gm_write_BEL( BEL, 'Sprinkler.uai.BEL')
>> br = gm_read_BEL( 'Sprinkler.uai.BEL', fg, rg )
br =
  1×1 cell array
    {1×6 cell}
```

Here is the file 'Sprinkler.uai.BEL'.

```
BEL
1
6 4 0.2500 0.2500 0.4500 0.0500 4 0.4000 0.1000 0.1000 0.4000 8 0.3500 0.0000 0.3150 ...
   ... 0.0350 0.1350 0.0150 0.0015 0.1485 2 0.5000 0.5000 2 0.7000 0.3000 2 0.5000 0.5000
```

6 Development

6.1 Tests

In gmtoolbox/TEST directory, Matlab functions allows to test the toolbox functions. They are non regression tests, exploring limit cases, example problems of the toolbox, and some interesting for development (see [3]).

A script TEST.m launches all tests.
A script Test_< $function$ >.m allows to test functions related to inference.
Some local functions generate specific problems for test.

The following table summarizes defined tests.

Factor graph	Tiny	Trivial	Lattice	Fig11	Fig4	Fig13	CHMM	DBN	Sprinkler
Test_rg_CVM.m gm_rg_CVM (regions)	x	x	x x x		x			x	
Test_rg_BETHE.m gm_rg_BETHE	x	x	x		x			x	
Test_rg_JT.m gm_rg_JT	x	x	x		x			x	
Test_include_evidence.m gm_include_evidence	x	x	x						
Test_infer_GBP.m gm_infer_GBP	CVM	CVM+ rg	CVM CVM+(x2) BETHE JT rg(x2)	CVM JT BETHE rg	CVM JT	CVM BETHE	CVM JT BETHE	CVM JT BETHE	CVM CVM+ rg

CVM+ stands for CVM with regions given in argument.
rg stands for regions graph given directly (not the output of a toolbox function).

Note that the script Test_infer_GBP_MAP.m checks the same tests as Test_infer_GBP.m but uses infer_GBP.m (other coding of GBP) instead of gm_infer_GBP.m.

6.2 Comparison with libDAI

6.2.1 Problems to test

We chose to use a function of the toolbox to generate a problem: gm_example_DBN. This function allows to create Dynamic Bayesian Networks, fixing the following parameters:

- v : number of variables in one time period,

- s : number of states of each variable,
- h : number of time period.

We call problems DBN-$< v > - < s > - < h >$.

6.2.2 Parametrization of libDAI

libDAI is launched with the command line.

We used the pre-defined parametrization, written in the file libDAI-0.3.2/tests/aliases.conf

```
GBP_MIN:   HAK[doubleloop=0,clusters=MIN,init=UNIFORM,tol=1e-9,maxiter=10000]
GBP_BETHE: HAK[doubleloop=0,clusters=BETHE,init=UNIFORM,tol=1e-9,maxiter=10000]
GBP_DELTA: HAK[doubleloop=0,clusters=DELTA,init=UNIFORM,tol=1e-9,maxiter=10000]
GBP_LOOP3: HAK[doubleloop=0,clusters=LOOP,init=UNIFORM,loopdepth=3,tol=1e-9,maxiter=10000]
GBP_LOOP4: HAK[doubleloop=0,clusters=LOOP,init=UNIFORM,loopdepth=4,tol=1e-9,maxiter=10000]
GBP_LOOP5: HAK[doubleloop=0,clusters=LOOP,init=UNIFORM,loopdepth=5,tol=1e-9,maxiter=10000]
GBP_LOOP6: HAK[doubleloop=0,clusters=LOOP,init=UNIFORM,loopdepth=6,tol=1e-9,maxiter=10000]
GBP_LOOP7: HAK[doubleloop=0,clusters=LOOP,init=UNIFORM,loopdepth=7,tol=1e-9,maxiter=10000]
GBP_LOOP8: HAK[doubleloop=0,clusters=LOOP,init=UNIFORM,loopdepth=8,tol=1e-9,maxiter=10000]
```

Documentation was find in:

- libDAI-0.3.2/doc/html/structdai_1_1HAK_1_1Properties.html
- source code libDAI-0.3.2/src/Hak.cpp et regiongraph.cpp (for constructCVM)

$booldai :: HAK :: Properties :: doubleloop$
Use single-loop (GBP) or double-loop (HAK)
The following cluster choices are defined:

- MIN minimal clusters, i.e., one outer region for each maximal factor
- DELTA one outer region for each variable and its Markov blanket
- LOOP one cluster for each loop of length at most Properties::loopdepth, and in addition one cluster for each maximal factor
- BETHE Bethe approximation (one outer region for each maximal factor, inner regions are single variables)

With BETHE, constructCVM is not called but is called with others.
$dai :: HAK :: Properties :: DAI_ENUM$ (InitType, UNIFORM, RANDOM)

$size_tdai :: HAK :: Properties :: loopdepth$

Depth of loops (only relevant for clusters == ClustersType::LOOP)
$Realdai :: HAK :: Properties :: tol$

Tolerance for convergence test.
$size_tdai :: HAK :: Properties :: maxiter$

Maximum number of iterations.

6.2.3 Compare execution time and beliefs

The execution time was measured on perdu PC.

In the following table, the time used to build region graph with gm_rg_CVM is not taken into account. To have an idea, it requires 0.3s for DBN-10-10-10 and 6s for DBN-10-10-40.

Execution time can vary. For example, when the program is launched consecutively, or if the C library version is changed (with an update of Ubuntu and a recompilation of libDAI). Several measures allow to check that the following times were representatives.
We used the script TEST/Test_DBN_libDAI.m which itself launches sh TEST/Test_DBN_libDAI.sh.

Problem DBN_ $< v >$ _ $< s >$ _ $< h >$	Execution time in s of GBP_MIN libDAI	Execution time in s of GBP Matlab (version May 17, 2018)	max(abs(difference) on beliefs
DBN-10-10-10	0.14	0.39	0.0016
DBN-20-10-10	0.30	0.76	0.0029
DBN-10-10-40	0.57	1.67	0.0018
DBN-10-10-60	0.79	2.80	0.0018
DBN-140-10-8	1.78	7.18	0.0019
DBN-144-10-10	2.06	10.31	0.0019

Bibliography

[1] D. Koller, N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2006.

[2] J. M. Mooij, *libDAI: A free & open source C++ library for Discrete Approximate Inference in graphical models* Journal of Machine Learning Research, 11(Aug):2169-2173, 2010.

[3] J. S. Yedidia, W. T. Freeman, Y. Weiss, *Constructing Free Energy Aproximations and Generalized Belief Propagation Algorithms*. IEEE Transactions on Information Theory, vol. 51, pp. 2282-2313, 2005.

[4] Eds. T. Morita, M. Suzuki, K. Wada, M. Kaburagi, Foundations and applications of cluster variation method ans path probability method. In Prog. Ther. Phys. Supplement, vol 115, 1994.

GMtoolbox functions

GM generation

gm_example_Sprinkler	Generate a Bayesian Network toy example
gm_example_Ecology	Generate a toy example related to ecology
gm_example_CHMM	Generate a Dynamic Bayesian Network
gm_example_DBN	Generate a Coupled Hidden Markov Model
gm_create_fg	Generate a factor graph
gm_separate_fg	Separate connected sub factor graphs

GMDP inference

gm_infer_GBP	Generalized Belief Propagation (parent-to-child)
gm_rg_CVM	Generate a region graph with Cluster Variation Method (CVM)
gm_rg_BETHE	Generate a region graph with BETHE approximation
gm_rg_JT	Generate a region graph such that GBP corresponds to Junction Tree (JT)
gm_include_evidence	Take into account observations
gm_plot_belief_evolution	Plot the evolution of belief for a range of GBP iteration

Utilities

gm_check_fg	Check the validity of a factor graph
gm_check_rg	Check the validity of a region graph
gm_check_rg_JT	Check if a region graph is of the type given by gm_rg_JT function
gm_plot_fg	Plot a factor graph
gm_plot_rg	Plot a region graph
gm_read_fg	Read a factor graph in UAI format
gm_read_evidence	Read evidences in UAI format
gm_read_MAR	Read marginals in UAI format
gm_read_BEL	Read beliefs of regions in UAI format
gm_write_fg	Write a factor graph in UAI format
gm_write_evidence	Write evidences in UAI format
gm_write_MAR	Write marginals in UAI format
gm_write_BEL	Write beliefs of regions in UAI format