The Belief Function Machine (BFM) An Environment for Reasoning with Belief Functions in MatLab

Phan H. Giang and Sushila Shenoy

University of Kansas School of Business
Summerfield Hall 1300 Sunnyside Ave.
Lawrence, KS 66045-7585, USA
phgiang@ku.edu – shenos2@rpi.edu

July 23, 2003

Contents

Part I

Backgrounds

This reference manual provides description of modules implemented by Phan Giang for the Belief Function Machine – an environment for reasoning with belief functions in MATLAB.

1 Belief function theory – A brief review

Belief function theory, also known as Dempster-Shafer theory after its founders or the Transferable Belief Model (TBM), is a framework to reason with generalized probabilities and/or with uncertainty provided by evidence. For an overview, the reader is referred to [**?**, **?**, **?**].

The main ingredients of this theory is the ways pieces of evidence are represented and combined. An evidence is embodied by a *basic probability (belief) assignment* (*bpa* or *bba*). Distinct pieces of evidence are combined by using Dempster's rule. A *bpa* is a function from the power set 2^S to the unit interval where S is the set of possible worlds

$$m : 2^S \to [0, 1] \text{ such that } \sum_{A \subseteq S} m(A) = 1$$

Value $m(A)$ for $A \subseteq S$ is called probability *mass* of A. A *focal element* or *focus* is a subset A for which probability mass is strictly positive $m(A) > 0$. It is important to note that while probability $p(A)$ is interpreted as the chance or belief that the true world happens to be one of states in A, mass $m(A)$ only conveys the support rendered by the piece of evidence at hand that the true world is in the set A. But this support could not be given to any proper subset of A. In particular, $m(\emptyset)$ could be viewed as a measure of inconsistency existed among available evidences. $m(S)$ is a measure of ignorance. Such interpreted, a condition that holds for probability, $p(A) \leq p(B)$ if $A \subseteq B$, does not hold for basic probability assignment.

From construct of basic probability assignment one could define several derivative constructs like *belief* function

$$Bel(A) \stackrel{\text{def}}{=} \sum_{B \subseteq A, B \neq \emptyset} m(B),$$

plausibility function

$$Pl(A) \stackrel{\text{def}}{=} \sum_{B \cap A \neq \emptyset} m(B)$$

and *commonality* function

$$Q(A) \stackrel{\text{def}}{=} \sum_{B \supseteq A} m(B)$$

$Bel(A)$ expresses the belief of the statement that the true world *is* definitely in set A while $Pl(A)$ is about the statement that the true world *could be* in set A. It is clear that $Pl(A) \geq Bel(A)$.

These are some forms in which a belief function can be represented. The relation between any two of those forms is one-to-one i.e., given a belief function in one form, it is possible to automatically calculate the other form. Strictly speaking, a belief function in a particular form is a mathematical entity. From now, by default, a belief function is assumed to be given in the mass form unless specifically said otherwise.

Given two belief functions m_1 and m_2 representing two distinct pieces of evidence, their combination is represented by another belief function (denoted by $m_1 \oplus m_2$) which is obtained from m_1, m_2 by Dempster's rule of combination

$$m_1 \oplus m_2(A) = \sum_{B \cap C = A} m_1(B).m_2(C)$$

In terms of variables, the set of possible worlds could be taken as the Cartesian product of the variable state-spaces $\Omega = \Omega_X \times \Omega_Y \times \ldots$ where $X, Y, \ldots$ are variables and their state-space ares $\Omega_X, \Omega_Y \ldots$

The set of variables on which a belief function is defined is called its *domain*. Two belief functions on different domains can be combined provided they have been *extended* to the join domain which is obtained by union of two separate domains. For example, suppose α, β are two sets of variables such that $\alpha \subseteq \beta$ and $\gamma = \beta - \alpha$. Suppose m is a belief function defined on domain α. A *vacuous* extension of m to β denoted by $m^{\uparrow \beta}$ is defined as

$$m(A) = p \Leftrightarrow m^{\uparrow \beta}(A \times \Omega_{\beta - \alpha}) = p$$

where $A \subseteq \Omega_\alpha$ and $\Omega_{\beta-\alpha}$ is the state-space of set of variables $\beta - \alpha$.

Operation *projection* is used if from a given belief function m on a domain (β) one interests in only a subset of variables in that domain (α). The result of this projection is a belief function denoted by $m^{\downarrow \alpha}$

$$m^{\downarrow \alpha}(A) = \sum_{B \subseteq A \times \Omega_{\beta - \alpha}} m(B)$$

2 Fusion algorithm for computing marginals

In many situations, reasoning with belief functions reduces to computing the marginal of a *join* on a set of variables. The join belief function is obtained by combination of belief functions representing all available evidence. Suppose these belief functions are $m_1, m_2 \ldots m_n$ and the set variables of interest is α then what need to be computed is

$$(m_1 \oplus m_2 \oplus \ldots m_n)^{\downarrow \alpha}$$

Since a brute force evaluation of this expression is computationally difficult, techniques based on *local computation* is investigated [**?**, **?**, **?**]. Shenoy proposes [**?**] the *fusion* algorithm for an efficient computation of marginals. This algorithm is implemented in the Belief Function Machine project.

Suppose β is the join domain of belief functions $m_1, m_2 \ldots m_n$, and α is the domain of the required marginal. The computation of the marginal can be viewed as a process of removing variables in $\beta - \alpha$ from the join domain β. An order of variables in $\beta - \alpha$ is selected. This order is called the *deletion sequence*. The variables are removed one by one according to the deletion sequence. The process of removing a variable X consists of the following steps

1. Combine all the belief functions that have X in its domain.
2. Project the combined belief function on the domain which is obtained by removing X.
3. Remove belief functions that have been used and add the new one into the pool of belief functions.

After all variables have been removed, a combination of the remaining belief functions yields the required marginal.

3 Representation of focal sets

The most costly component in this process is the combination operation, which also includes the extension operation if necessary. For an efficient implementation of combination, the choice of representation of focal sets is crucial. A focal set is a subset of the state-space. There are *four* representation

for it. Suppose we are working with a belief function whose domain has 3 variables X, Y, Z and the cardinalities of their state-spaces are respectively 2, 3 and 10. For example, let us consider a focal set $A = \{x_2y_1z_1, x_2y_2z_4, x_1y_3z_5, \}$.

Set form: A focal set is represented by a matrix. Each row is an element of the set. Each column corresponds to a variable. The focal set A is

$$\begin{bmatrix} 2 & 1 & 1 \\ 2 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

Bit vector: Each focal set corresponds to a bit vector of length that equals the size of join domain state-space. Each position corresponds to an element of the set. The value at position i is 1 if the corresponding element is present in the focal set. Otherwise, it is 0. For example the bit vector for A is
01000000000000000000010000001000000000000000000000000000000000
Notice, for example, that element $x_2y_2z_4$ occupies position number 22, which is calculated by

$$2 + (2 - 1) * 2 + (4 - 1) * 6 = 22$$

Integer vector: This representation is obtained from bit vector by splitting it into segments of fixed length (by default 50, except the last segment) and converting the bit strings into decimal numbers. For example, A is represented by

[281475247243264 0]

Coordinate form A focal set is represented by a n-D matrix M where n is the number of variables in the domain such that $M(i, j, k, \ldots) = 1$ iff element $x_iy_jz_k \ldots$ is in the focal set. Otherwise, it is 0. For example, focal set A is represented by 3-D matrix of 10 pages, 3 columns and 2 rows as follows

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{\text{page } 1} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{4} \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{5}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Although these forms are different, they all are necessary because each is suitable for a specific purpose. For example, for a human being, the set form is most readable, while to store a belief function the integer vector form is most economical. The coordinate and bit vector forms are convenient for manipulating belief functions. Given information about the variables (names, cardinalities of state-spaces) it is possible to convert one form to another.

4 UIL language

UIL stands for the *Unified Input Language* is used to describe the input data of a problem. We present the UIL by two examples.

4.1 A Captain Problem

Problem description

This example is borrowed from *Russell Almold, Graphical Belief Modeling, Chapman & Hall 1995.*

A ship captain is concerned about the *arrival delay* (A) i.e., how many days his ship will arrive late to the destination port. The arrival delay is the sum of *departure delay* (D) and *sailing delay* (S). There are three factors influencing value of departure delay: *Loading delay* (L), *Maintenance delay* (M) and *Forecast of foul weather* (F). If true each variable adds one day to departure delay. The sailing delay depends on two factors: the need for a *Repair* at sea (R) and the foul *Weather*.

For the captain, variables L, F, M, W, R are binary variables i.e., they can accept one of two values denoted by $\{t, f\}$. From that he estimate that sailing and departure delays are integral numerical variables that vary from 0 to 3 days. Therefore, arrival delay varies from 0 to 6 days. The defined variables are summarized as follows

Var.	Description	Frame	Notes
A	Arrival delay	$\{0, 1, \ldots, 5, 6\}$	days
D	Departure delay	$\{0, 1, 2, 3\}$	days
S	Sailing delay	$\{0, 1, 2, 3\}$	days
L	Loading delay	$\{t, f\}$	t: there is delay
F	Forecast	$\{t, f\}$	t: forecast of foul weather
M	Maintenance	$\{t, f\}$	t: maintenance at dock
W	Weather	$\{t, f\}$	t: actual foul weather
R	Repair	$\{t, f\}$	t: repair made at sea

The captain knows the following

- Each of variables the Weather or Repair at sea if true adds 1 day to sailing delay. This proposition is true 90% of the time.
- The weatherman is reliable about 80% of the time. That is the matching between Forecast and Weather occurs 80%.
- Given the ship has undergone a maintenance at dock, the chance it needs a repair at sea is between 10% to 30%.
- Given the ship has not undergone a maintenance, the chance it needs a repair at sea is between 20% to 80%.
- Forecast predicts 20% chance of a foul weather, and 60% chance of a fair weather.
- There is 30% chance that loading will be delayed and 50% chance that loading is on schedule.
- The ship does not undergo maintenance at dock.

Coding in UIL language

The following is the print out of file *captain.txt*.

```
# Captain's decision problem
# Russell Almold, Graphical Belief Modeling, Chapman & Hall 1995

# Variable A denotes Arrival delay
# Variable D denotes Departure delay
```

```
# Variable S denotes Sailing delay
# Variable L denotes Loading delay
# Variable F denotes Forecast of weather
# Variable M denotes Maintenaince performed
# Variable W denotes Weather
# Variable R denotes Repair made

DEFINE VARIABLE A {0 1 2 3 4 5 6}; # Arrival delay is between 0 to 6 days
DEFINE VARIABLE D {0 1 2 3}; # Departure delay is between 0 to 3 days
DEFINE VARIABLE S {0 1 2 3}; # Sailing delay is between 0 to 3
DEFINE VARIABLE L {t f}; # Loading can be delayed or not
DEFINE VARIABLE F {t f}; # Forecast weather can be either foul or fair
DEFINE VARIABLE M {t f}; # Maintenance is either done or not
DEFINE VARIABLE W {t f}; # Weather is either foul or fair
DEFINE VARIABLE R {t f}; # Repair is either done or not

DEFINE RELATION ADS {A D S};
DEFINE RELATION DLMF {D L M F};
DEFINE RELATION SWR {S W R};
DEFINE RELATION WF {W F};
DEFINE CONDITIONAL RELATION RM1 {R} GIVEN {M};
DEFINE CONDITIONAL RELATION RM2 {R} GIVEN {M};
DEFINE RELATION PRIORF {F};
DEFINE RELATION PRIORL {L};
DEFINE RELATION PRIORM {M};

# The arrival delay is sum of departure and sailing delays
SET VALUATION ADS {
(0 0 0) (1 1 0) (2 2 0) (3 3 0)
(1 0 1) (2 1 1) (3 2 1) (4 3 1)
(2 0 2) (3 1 2) (4 2 2) (5 3 2)
(3 0 3) (4 1 3) (5 2 3) (6 3 3) } 1;

# Loading delay, foul weather forecast and maintenance if true
# adds one day each to the departure delay
SET VALUATION DLMF {
(0 f f f) (1 t f f) (1 f t f) (1 f f t)
(2 f t t) (2 t f t) (2 t t f) (3 t t t) } 1;
```

```
# Heavy weather and repair at sea adds 1 day to the sailing delay.
# This proposition is true about 90% of the time
SET VALUATION SWR {
(0 f f) (1 t f) (1 f t) (2 t t)} .9;

# The weatherman is reliable about 80% of the time
SET VALUATION WF {(f f) (t t)} .8;

# Given the ship has undergone a maintenance, the chance it needs a repair
# at sea is between 10% to 30%
SET CONDITIONAL VALUATION RM1 GIVEN {t}, {t} .1, {f} .7;

# Given the ship has not undergone a maintenance, the chance it needs a repair
# at sea is between 20% to 80%
SET CONDITIONAL VALUATION RM2 GIVEN {f}, {t} .2, {f} .2;

# Forecast predicts foul weather with chance .2, fair weather with chance .6
SET VALUATION PRIORF {t} .2, {f} .6;

# Loading is delayed with chance .3 and on schedule with chance .5
SET VALUATION PRIORL {t} .3, {f} .5;

# The ship is not maintained
SET VALUATION PRIORM {f} 1;

# Question of interest to the Captain: Belief about Arrival delay.
```

Explanatory note

A UIL file is a *text* file which can be prepared in any text processor. Several commands allow a user to declare input data.

1. A commentary begins with sign #. UIL translator ignores the rest of the line after #. Commentaries are often placed in a separate line or at the end of a command line. For example `# Captain's decision problem`.

2. A variable definition starts with the key word

`DEFINE VARIABLE` in upper case
followed by a *variable name* and then the *frame* for the variable. A command ends with a semicolon (;). A variable name is a string of characters beginning with a letter. In print-out only first 5 characters are used. A variable frame is a list of names separated by a blanks and enclosed in curly braces { }. For example,
`DEFINE VARIABLE A {0 1 2 3 4 5 6};`

3. A relation definition begins with the key word
   `DEFINE RELATION`
   followed by a relation name and then the relation's *domain* i.e, the list of variables involved in the relation. The variables in the list must be defined. Variable names are separated by blanks and are enclosed in curly braces. For example
   `DEFINE RELATION ADS {A D S};`

4. A conditional relation definition begins with the key word
   `DEFINE CONDITIONAL RELATION`
   followed by the name of the (conditional) relation, the relation's domain and then the conditioning domain i.e., the list of variables. For example,
   `DEFINE CONDITIONAL RELATION RM1 {R} GIVEN {M};`

5. A relation valuation input begins with key word
   `SET VALUATION ADS`
   followed by a relation name and the list of foci and their masses separated by comma (,). Each focus is a list of elements that are separated by blanks and are enclosed in curly braces. Each element is a list of variable values in the order according to the relation definition separated by blanks. For example,
   `SET VALUATION WF {(f f) (t t)} .8, {(f t) (t f)} .2;`
   The masses are supposed to add up to 1. A sum greater than 1 is invalid. When the sum is less than 1, the remaining mass is assigned to the whole frame (tautology).

6. A conditional valuation starts with the key word
   `SET CONDITIONAL VALUATION` followed by the conditional relation name, the value of conditioning variable and then list of foci

and their masses. For example
`SET CONDITIONAL VALUATION RM2 GIVEN {f}, {t} .2, {f} .2;`

Note also that upper and lower ends of a probability interval are translated into belief functions framework as plausibility and belief. For example "given the ship has undergone a maintenance at dock, the chance it needs a repair at sea is between 10% to 30%" is coded as
`SET CONDITIONAL VALUATION RM1 GIVEN {t}, {t} .1, {f} .7;`.

4.2 A Chest Clinic problem

Problem description

The following is the print out of *lschestclin.txt* file. It encodes a Lauritzen - Spiegelhalter's Chest Clinic problem which is often represented as a Bayesian net. It shows that reasoning with Bayesian nets can be done using BFM (although BFM is not a best tool for handling a Bayesian net).

```
# Lauritzen and Spiegelhalter's Lung Cancer Diagnosis problem

DEFINE VARIABLE A {a ~a};  # Visit to Asia
DEFINE VARIABLE S {s ~s};  # Smoking
DEFINE VARIABLE T {t ~t};  # The patient either has Tuberculosis or not
DEFINE VARIABLE L {l ~l};  # Lung cancer
DEFINE VARIABLE B {b ~b};  # Bronchitis
DEFINE VARIABLE E {e ~e};  # Either T or B
DEFINE VARIABLE X {x ~x};  # Positive X-ray
DEFINE VARIABLE D {d ~d};  # Dyspnea

# Tuberculosis given visiting Asia
DEFINE CONDITIONAL RELATION TA1 {T} GIVEN {A};
DEFINE CONDITIONAL RELATION TA2 {T} GIVEN {A};

# Lung cancer given smoking
DEFINE CONDITIONAL RELATION LS1 {L} GIVEN {S};
DEFINE CONDITIONAL RELATION LS2 {L} GIVEN {S};

# Bronchitis given smoking
DEFINE CONDITIONAL RELATION BS1 {B} GIVEN {S};
```

```
DEFINE CONDITIONAL RELATION BS2 {B} GIVEN {S};

# Model either Tuberculosis or Lung cancer
DEFINE RELATION ETL {E T L};

# Positive X-ray given Either T or L
DEFINE CONDITIONAL RELATION XE1 {X} GIVEN {E};
DEFINE CONDITIONAL RELATION XE2 {X} GIVEN {E};

# Dyspnea given B and Either (T or L)
DEFINE CONDITIONAL RELATION DBE1 {D} GIVEN {B E};
DEFINE CONDITIONAL RELATION DBE2 {D} GIVEN {B E};
DEFINE CONDITIONAL RELATION DBE3 {D} GIVEN {B E};
DEFINE CONDITIONAL RELATION DBE4 {D} GIVEN {B E};

DEFINE RELATION PRIORS {S};
DEFINE RELATION PRIORA {A};

# Given Asia, the chance of T is 5%
SET CONDITIONAL VALUATION TA1 GIVEN {a} {t} .05 {~t} .95;

# If not, the chance is 1%
SET CONDITIONAL VALUATION TA2 GIVEN {~a} {t} .01 {~t} .99;

# Lung cancer given smoking is 10%
SET CONDITIONAL VALUATION LS1 GIVEN {s} {l} .1 {~l} .9;

# If not smoking, chance of L is 1%
SET CONDITIONAL VALUATION LS2 GIVEN {~s} {l} .01 {~l} .99;

# Bronchitis given smoking is 60%
SET CONDITIONAL VALUATION BS1 GIVEN {s} {b} .6 {~b} .4;

# Bronchitis given not smoking is 30%
SET CONDITIONAL VALUATION BS2 GIVEN {~s} {b} .3 {~b} .7;

# E if and only if either T or L
SET VALUATION ETL {(e t l) (e t ~l) (e ~t l) (~e ~t ~l)} 1;
```

```
# Given Either T or L, X-ray is positive 98% of the time
SET CONDITIONAL VALUATION XE1 GIVEN {e} {x} .98 {~x} .02;

# If neither T nor L, X-ray is positive 5% of the time
SET CONDITIONAL VALUATION XE2 GIVEN {~e} {x} .05 {~x} .95;

# Given both b and e, the chance of d is 90%
SET CONDITIONAL VALUATION DBE1 GIVEN {b e} {d} .9 {~d} .1;

SET CONDITIONAL VALUATION DBE2 GIVEN {b ~e} {d} .7 {~d} .3;

SET CONDITIONAL VALUATION DBE3 GIVEN {~b e} {d} .8 {~d} .2;

SET CONDITIONAL VALUATION DBE4 GIVEN {~b ~e} {d} .1 {~d} .9;

SET VALUATION PRIORS {s} .5 {~s} .5;
SET VALUATION PRIORA {a} .01 {~a} .99;
```

Explanatory note

There are two points should be noted.

1. Each conditional probability potential (table) corresponds to a conditional relation. For example, probability of Dispnea is conditioned on two variable Bronchitis and E. For each combination of values of B and E, there is a probability table for D. Because each B and E are binary, therefore, four relations to be defined.

   ```
   # Dyspnea given B and Either (T or L)
   DEFINE CONDITIONAL RELATION DBE1 {D} GIVEN {B E};
   DEFINE CONDITIONAL RELATION DBE2 {D} GIVEN {B E};
   DEFINE CONDITIONAL RELATION DBE3 {D} GIVEN {B E};
   DEFINE CONDITIONAL RELATION DBE4 {D} GIVEN {B E};
   ```

   The conditional relation $DBE1$ corresponds to $P(\cdot|b, e)$, $DBE2$ to $P(\cdot|b,\ e)$, $DBE3$ to $P(\cdot|\ b, e)$, and $DBE4$ to $P(\cdot|\ b,\ e)$.

2. Foci of probability relation are all singletons. One has to make sure that the masses add up to 1. Because otherwise the un alloted mass would be assigned to the tautology making the relation no longer probability. For example
SET CONDITIONAL VALUATION DBE4 GIVEN {~b ~e} {d} .1 {~d} .9;

5 Examples of using BFM

In this section, we present examples of how to use BFM. This is given as in the forms of batch files applied on the Captain decision problem and Chest clinic.

5.1 Captain decision problem

```
uil2bm('uilfiles\captain.txt', 'bmcaptain')
% translate UIL into BFM format

global BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME QUERY BELTRACE
global NODE BJTREE TRANPROTOCOL;
% declaration of global variables

load bmcaptain
% load file containing BFM data structure

embelall = condiembed([BELIEF(:).number]);
% convert into unconditional beliefs

keepbel(embelall);
% clear unnecessary beliefs from memory

showbel(embelall)
% show the input in unconditional form

showbel(solve(embelall, 'A'))
% calculate then show the marginal on Arrival delay

showbel(4)
```

```
% show belief on Weather and Forecast

aaa = [.8 .7 .6 .5; .2 .3 .4 .5];
% matrix reflecting variation of belief masses

showbel(senanalysis(4, aaa, 'A'))
% do sensitivity analysis then show marginal on Arrival delay

bjtbuild('A', embelall)
% construct a binary join tree with variable A as final node

showbel(solvetreeall(1))
% calculate marginal for all variables based on BJTree and show

showbel(7)
% show the Forecast

bbb = [.6 .55 .5 .45; .2 .2 .2 .2; .2 .25 .3 .35];
% a matrix of masses reflecting different Forecast

showbel(senanalysis(7, bbb, 'A', embelall))
% do sensitivity analysis on A depending on different F
```

5.2 Chest clinic problem

```
uil2bm('uilfiles\lschestclin.txt', 'bmlschest')
% translate UIL input into BFM data structure

global BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME QUERY
global BELTRACE NODE BJTREE TRANPROTOCOL;
% declare global var

load bmlschest
% load BFM data for the example

emallbels = condiembed([BELIEF(:).number]);
% convert conditional belief into unconditional ones
```

```
keepbel(emallbels);
% clear the memory of unncessary structures

showbel(emallbels)
% show all beliefs for visual examination

showbel(solve(emallbels, 'T'))
% show marginal for Tuberculosis

bjtbuild('L', emallbels)
% build a binary join with L as the final node

showbel(solvetreeall(1))
% calculate marginals for each individual variables

belupdate(1, 15, observe('A', 'a'))
% add observation the patien visits Asia

showbel(solvetreeall(1))
% recalculate marginals for all individual variables
```

6 Module description format

In the next part we catalog the modules found in the following format:

modulename

Synopsis: One line description of module usage.

Input:

- input #1
- input #2

Output: the output of the module

Algorithm description:

Step 1 What does the first segment of the code do.

Step 2 What does the next segment of the code do.

Uses: What modules are used in this module.

Used in: In what modules this module is used.

see also Oher modules that perform the related functions.

Part II

Catalog of Modules

1 approx.m

Synopsis: APPROX does approximation for the beliefs. It removes the foci with masses less than a threshold and lumps those masses to the one mass assigned to the union of removed foci. It leaves a belief with at most 1 focus with mass less than the threshold.

Input: `inbelidlist` is a list of belief id numbers. `threshold` is the threshold foci with masses lower than that is lumped together.

Output: `appbelidlist` is the list of belief id numbers created.

Algorithm description:

Global `BELIEF VARIABLE FRAME`

Check The validity of the input is checked. If there are beliefs with the id-numbers given by the input and if these beliefs exist in m-form. This function requires that the belief to be approximated has only one vector of masses. If the input is invalid, the computation stops.

Identify and mark the foci with masses less than the given threshold. If there is more than one of such then go to next step.

Take the sum of the small masses. Create the new focus by taking the union of the marked foci.

Remove the marked foci (with small masses). Add the new focus to the list of foci. Remove the small masses from the mass vector and append their sum to the vector.

Sort list of foci by their masses.

Create a new belief

Uses: `local_unionfoci`, `collapsemat`

Used in:

see also `ultco`bine.m+, `bproject.m`

2 bcombine.m

Synopsis: BCOMBINE combines two belief potentials in $m-$form.

Input: `bel_id1, bel_id2`: are two belief numbers.

Output: `bel_id3`: is the id-number of the newly created belief.

Algorithm description:

Global `BELIEF VARIABLE MOFI ATTRIBUTE, STRUCTURE FRAME`

Check The validity of the input is checked. If there are beliefs with the id-numbers given by the input and if these beliefs exist in m-form. If the input is invalid, the computation stops.

Conversion focal sets. Each focal set stored in BELIEF is an integer vector. Each integer (or word) except the last one in the vector represents 50 (the value given by `word_size`) bits of the bit vector representation of the focal set. The number of useful bits in the last word is calculated by taking remainder of frame size (`pframe1_size`) divided by 50. The matrix of bit vector representing focal sets is then *reshaped* into $(n+1)$-dim array where each of first n dimensions corresponds to a variable in the domain of the belief function. The last dim is used for number of focal sets. The result is contained in `crdn1_foset`.

Rearrange the order of variables in belief domains is done by *permute* command. After rearrangement, each domain has 2 segments. The first segment consists of variables common to both domains. The second segment consists of variables proper for each domain.

Extension of domains into the join domain is done by command *repmat*. A number of cases has to be handled separately. After extension, the arrays need to be rearranged so that the order of variables in both *extended* domains is the same.

Conversion to integer matrix. Each bit array is then converted into an integer matrix.

Intersection of focal sets is done by *bitand* command.

The join focal sets array (`intersection_foset`) is concatenated with vector represents product of masses (`fin_value`).

The obtained matrix is then *collapsed* to remove identical focal sets. After that it is *sorted* by a descending order of mass.

A new belief is created and a new birth record of the belief is generated.

Normalize if required.

Approximate if required.

Uses: `varreference`, `extfind`, `int2bitv`, `bitmat2intmat`, `collapsemat`, `birthrecord`.

Used in:

see also `ultco`bine.m+, `bproject.m`

3 bel2prob.m

Synopsis: BEL2PROB transforms mass values into probability. Two transformations are used: pignistic and plausibilistic.

Input: `bellist` the list of belief id numbers.

Output: `resultbel` the list of belief id numbers.

Algorithm description:

Convert	integer vector representation of foci of each belief into bit vector form.
Identify	the singletons that belongs to at least one focus.
Calculate	masses to be assigned to the singletons.
Normalize	masses into probability vectors.

Uses: `extfind.m`, `int2bitv`

Used in:

4 bel2var.m

Synopsis: BEL2VAR gives the list of variables that are included in the domains of beliefs whose numbers are in `bel_list`.

Input: `varbelstruct`, `bel_list`

- `varbelstruct` is the structure that contains information about variable-belief relationship. This structure has 5 fields
  1. `varbelstruct.varnums`: the list of variable id numbers
  2. `varbelstruct.varcard`: the list of variable cardinalities
  3. `varbelstruct.belnums`: the list of belief id numbers
  4. `varbelstruct.belcard`: the list of belief cardinalities (numbers of focal sets)
  5. `varbelstruct.cross`: the table of cross reference
- `bel_list` is a list of belief id numbers.

Output: `BEL_LIST` the list of variable id numbers.

Algorithm description:

Find positions of beliefs in the vector of all belief numbers

Find positions of variables involved. Then output the list their id numbers.

Uses: `extfind.m`

Used in:

see also `ar2bel.m+`

5 beladd.m

Synopsis: BELADD adds beliefs in `bels` to a var-bel structure and outputs a new var-bel structure.

Input:

- `invarbel` is a var-bel structure with 5 fields (`varnums, varcard, belnums, belcard`, and `cross`).
- `bels` is the list of belief numbers to add into var-bel structure.

Output: `varbel` is the updated var-bel structure.

Algorithm description:

Global `BELIEF`

Check if some belief numbers in `bels` are already present in `invarbel`. If so exclude them from the list to add.

Add For each belief to be added: retrieve its information from BELIEF structure (domain, number of focal sets). This information for all beliefs in `bels` is put in a matrix `toaddvarbel` whose column is a bit vector telling which variables are present in the domain of the belief. Vector `toaddbel_card` tells the number of focal sets of each belief.

Uses: `extfind.m`

Used in:

see also `aradd.m+,` \erb+belremove.m+

6 belremove.m

Synopsis: BELREMOVE removes beliefs in `bels` from a var-bel structure and outputs new var-bel structure.

Input:

- `invarbel` is a var-bel structure with 5 fields (varnums, varcard, belnums, belcard and cross)
- `bels` is a list of beliefs to be removed from `invarbel`

Output: `varbel` is the updated var-bel structure.

Algorithm description:

Remove beliefs from the variable-belief structure.

Remove variables that are not present in any domain of remained beliefs.

Uses:

Used in:

see also `arremo`e.m+, `beladd.m`

7 belupdate.m

Synopsis: BELUPDATE updates affected beliefs in binary join tree after one input belief has been changed into another belief.

Input:

- `tree` the id number of a binary join tree.
- `oldbel` the id number of belief to be changed.
- `newbel` id number of the new belief. `oldbel` and `newbel` must share the same domains (as sets) or otherwise the structure of tree would not be the same.

Output: `affectedbel` the list of id numbers of updated beliefs.

Algorithm description:

Global	`BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME QUERY BELTRACE NODE BJTREE`
Check	consistency conditions.
Identify	the node to which the old belief is attached.
Clear	messages out-going from the node and its children.
Update	the node with new belief number.
Recalculate	out-going messages.

Uses: `clearmsge, extfind, findneighbor, messagepass`

Used in:

8 birthrecord.m

Synopsis: BIRTHRECORD keeps a birth record of a belief item in the BELIEF structure. The records are kept in structure BELTRACE, which has 4 fields

1. `BELTRACE().number`: the record id number
2. `BELTRACE().action`: the action that creates the output belief (1: projection, 2: combination)
3. `BELTRACE().bel_ins`: the list of input belief id numbers
4. `BELTRACE().bel_out`: the id number of the belief produced.

Input:

- `action` is the action that creates the belief. This value is 1 if the belief is obtained from a projection, 2 if it is created by a combination.
- `bel_inputs` is the id numbers of parent beliefs. For projection, it consists of a single belief id number. For combination, it is a list of 2 belief id numbers.
- `bel_output` is the id number of child belief.

Output: `record_id` is the id number of the record

Algorithm description:

Global `BELTRACE`

Create a record of belief.

Uses:

Used in:

9 bitmat2intmat.m

Synopsis: BITMAT2INTMAT converts bit vectors into integer matrix. Each integer in a row is converted from a fixed number of bits except the last integer in a row.

Input:

- `bitvmat` is a bit matrix
- `word_size` is the number of bits used to make an integer (default value is 50)

Output: `INTMAT` is a matrix where each row is an integer representation of a bit vector.

Algorithm description:

Calculate	the size of the last word of each row. Then patch additional space so that the last words have the regular size.
Reshape	the bit matrix into one in which each row has a length of `word_size`,
Convert	the reshaped matrix into integer matrix.

Uses:

Used in:

10 bitstrinverse.m

Synopsis: BITSTRINVERSE converts a matrix of char '0' and '1' into matrix of char in which `'0' --> '1'` and `'1' --> '0'`.

Input: `bitstrmat` is bit string matrix.

Output: `invbitstrmat` is inverted bit string matrix.

Algorithm description:

Inverse each of character.

Uses:

Used in:

11 bjtbuild.m

Synopsis: BJTBUILD constructs binary join tree from BELTRACE that records fusion process.

Input:

- `avariable` is a variable whose marginalization provides trace of beliefs that will be assembled into a bjtree.
- `listbels` list of beliefs from those fusion process and the tree is constructed.

Output: `bjtidnum` is the id number of a binary join tree.

Algorithm description:

Check if a query that starts with the given variable exists. In case it does not, create such a query using `solve`.

Trace the belief creations using `local_traceback`. The result is the vector of `BELTRACE` records.

Construct a binary join tree from the belief trace. That includes construction of a list of nodes of the tree, a connection matrix, two matrices on message passing: One for messages that go from a node in row to a node in column, the other from column to row.

Uses: `local_traceback, solve, extfind`

Used in:

12 bproject.m

Synopsis: BPROJECT projects a belief on a set of variables

Input:

- `bel_id1` is the id number of the belief to be projected.
- `prj_domain` is the list of variable id numbers - the project domain.

Output: `bel_id2` is the id number of project belief.

Algorithm description:

Global `BELIEF`

Check the validity of `bel_id1` as a id number of an existing belief.

Calculate the size of the last word (number of bits used).

Convert integer representation of focal sets into bit vector by function `int2bitv` and then *reshape* bit array into the coordinate form.

Project on the required domain.

Convert bit array into integer array by function `bitmat2intmat`

Attach the masses with the focal sets and sum the masses assigned to the same focal set. Then sort the focal set by mass.

Create a new belief and a birth record.

Uses: `birthrecord.m`, `bitmat2intmat.m`, `collapsemat.m`, `extfind.m`, `int2bitv.m`.

Used in:

see also `com`ine.m+.

13 bvec2set.m

Synopsis: BVEC2SET converts the bit vector representation of foci into coordinate representation.

Input:

- `bitvset` is a 2-D matrix in which each row is a bit vector (char '0' or '1') representation of a focal set.
- `vars` is a structure containing 2 components.
  1. vars.nums: a vector of variable id numbers
  2. vars.card: a vector of cardinalities of variable frames.

Output: `crdnfoset` is a structure containing 2 fields

1. `crdnfoset.domain`: a vector variable id numbers
2. `crdn.indices`: a 2-D array where

   Each row represents a coordinates of one element of a focal set

   A row consists of -1s tells the end of a focal set

   A row of 0s denotes the empty set.

Algorithm description:

Check the consistency of the input data (the presence of char other than 1 or 0, the length of bit vector and the size of frame, presence of the empty set as focal set)

Extract the serial position of element 1s.

Convert the serial position into coordinates.

Uses:

Used in:

see also `nt2btv.m+`, `set2intmat.m`, `set2vec.m`.

14 chckcomb.m

Synopsis: CHCKCOMB checks correctness of combination implementation against a number of properties must be satisfied: commutativity and associativity.

Input: `belid1, belid2, belid3` are belief id numbers.

Output: `errormat` is a matrix that records the instances which cause error.

Algorithm description:

Global `BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME`

`QUERY BELTRACE NODE BJTREE`

Check the number of input arguments. If there are two inputs, check commutativity. If there are three inputs, check associativity.

Case 1 (2 inputs). From each input, generate a list of beliefs by changing the order of variables in its domain. Combine pairwise on the two generated lists. Compare the results of combinations.

Case 2 (3 inputs). Perform combination by each of the of three possible order of three-way combination ((12)3), ((23)1) and ((13)2). Compare the results.

Uses: `extfind, local_belperm, bcombine, isbelequal`

Used in:

15 chngdomorder.m

Synopsis: CHNGDOMORDER changes the order of variables in the domain of a belief.

Input:

- `inbelid` is a belief id number.
- `varorder` is the vector of variable id numbers.

Output: `belid` is a new belief id number.

Algorithm description:

Global `BELIEF VARIABLE FRAME`

Check equality between the set of variables in the belief domain and the set of variables in the order vector.

Change the order of variables according to the given order.

Uses: `extfind, int2bitv`

Used in:

16 chooseaseq.m

Synopsis: CHOOSEASEQ selects a
deletion sequence for the
fusion algorithm from a set of variables to be deleted.

Input:

- `varbelstruct`: variable-belief structure that has 5 fields
  1. `varbelstruct.varnums`
  2. `varbelstruct.varcard`
  3. `varbelstruct.belnums`
  4. `varbelstruct.belcard`
  5. `varbelstruct.cross`
- `invar_vec`: a vector of variables to be deleted.

Output: `var_vec` the vector of variables given in the deletion order.

Algorithm description:

Variables are sorted by number of beliefs involved.

Uses: `extfind.m`

Used in:

see also `hoosetwobel.m+`

17 choosetwobel.m

Synopsis: CHOOSETWOBEL selects two from a list of beliefs for combination.

Input: `varbelstruct` is the variable-belief structure with 5 fields

1. `varbelstruct.varnums`
2. `varbelstruct.varcard`
3. `varbelstruct.belnums`
4. `varbelstruct.belcard`
5. `varbelstruct.cross`

Output: `twobeliefs` is a vector of length 2 containing two belief id numbers.

Algorithm description:

The first belief is determined by the first belief listed in the `varbelstruct`.

The second belief is selected so that the combined frame has the minimal cardinality (state space).

Uses:

Used in:

see also `hooseaseq.m+`

18 clearbelnode.m

Synopsis: CLEARBELNODE clear all beliefs and nodes which are not attached to the tree. This function uses to

free memory from records that are not directly related to a binary join tree.

Input: `tree` is a binary join tree id number.

Output: `result` result = 1 means OK; result = 0 means abnormal end.

Algorithm description:

Global `BELIEF BELTRACE NODE BJTREE`

Identify the nodes of a given tree.

Identify the beliefs that attached to the nodes.

Clear beliefs and nodes not related to the bj tree.

Uses: `extfind`

Used in:

19 clearmsge.m

Synopsis: CLEARMSGE clear all
messages in a tree outgoing from a node.

Input:

- `tree` is a bjtree id number.
- `startnode` is a node id number.

Output: `result` result = 1 means OK; result = 0 means abnormal end.

Algorithm description:

Global `NODE BJTREE`

Take the start node as the root of a given tree.

Clear messages out-going from the root. Identify its children and put in a list.

Clear messages out-going from the nodes in the list. Identify their children.

Uses: `extfind`

Used in:

20 code2label.m

Synopsis: CODE2LABEL converts focal sets in which element is represented by its code into one of
labels.

Input:

- `crdnfoset` – a integer matrix
- `varlist` – a list of variables

Output: `columlabels` is a cell array of char matrices.

Algorithm description:

Global `BELIEF FRAME VARIABLE`

Check if the number of variables equals the number of columns in input matrix.

Read the frames that associate with the variables.

Process column by column. Numbers in a column are replaced by their labels in `tmp_label` and then put into a cell.

Uses: `extfind.m`

Used in: `showbel.m`

21 collapsemat.m

Synopsis: COLLAPSEMAT purges repeated rows in 2-D `inmatrix`, leaves only one representative so that all rows in purged matrix are different. It adds values in `value_column` of the removed rows into value of the representative.

Input:

- `inmatrix` is a 2-D array in which one column is designated as value column.

  Other columns are designated as address.

- `value_column` is the position number of the value column. By default the value column is the last one.

Output: `outmatrix` is the purged matrix.

Algorithm description:

Identify the value column and rearrange the matrix so that the value column is the last one.

Process two consecutive rows at a time. If they are identical in address part, then the number in value column of the second row is added into the value number of the first row. The value for the second row is set to 0.

Remove the rows having 0 as value.

Uses:

Used in: `bcombine.m`, `bproject.m`

22 condiembed.m

Synopsis: CONDIEMBED converts a
conditional belief into unconditional belief using
conditional embedding (Smets' rule)

Input: `inbelidlist` a list of id numbers for input (conditional) beliefs.

Output: `belidlist` the list of id numbers for new transformed (unconditional) beliefs.

Algorithm description:

Global `BELIEF VARIABLE FRAME`

Check if a belief in the list is conditional one.

Expand the domain to the union of conditioned domain and conditioning domain.

Expand each focus to the expanded domain.

Uses: `extfind, int2bitv, bitstrinverse, birthrecord`

Used in:

23 csolvetree.m

Synopsis: CSOLVETREE

find marginal on a list of variables when the potential attached to node is added by a new belief potential.

Input:

- `tree` is a binary join tree number
- `updatenode` is a node where potential will be updated by adding new belief.
- `addedbel` is the new belief to be added.
- `listvar` is the list of variables on which marginal is requested.

Output: `outputbel` is the id number of calculated marginal.

Algorithm description:

Global `BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME`

`QUERY BELTRACE NODE BJTREE`

Check consistency conditions: tree number, node number, belief number and relationship of node and belief.

Clear the messages affected by change in the update node.

Update belief attached to the node by the combination of existing belief with newly added belief.

Uses: `bcombine, clearmsge, extfind`

Used in:

24 discount.m

Synopsis: DISCOUNT
discounts beliefs by a rate.

Input:

- `inbelidlist` is a list of id numbers for input beliefs.
- `discountvec` is a vector of rates to which discount is made. If there is only one rate. Then this rate applies for all beliefs in the list.

Output: `belidlist` is the list of new belief id numbers.

Algorithm description:

Global `BELIEF VARIABLE FRAME`

Check consistency conditions: tree number, node number, belief number and relationship of node and belief.

Check if the tautology exists among foci for each belief. Create a tautology focus if it does not already exist. Assign a mass equal to the discount rate.

Update the masses assigned to each focus by the discount rate.

Uses: `birthrecord, bitmat2intmat, extfind`

Used in:

25 extfind.m

Synopsis: EXTFIND find the indices (position) of each element of a vector (`search_list`) in another vector (`target_list`)

Input:

- `search_list` is the searching vector.
- `target_list` is the searched vector.

Output: `indxvec` contains the positions of elements in `search_list` with respect to the second list `target_list`. If an element does not exist in `target_list`, then its position is set to 0.

Algorithm description:

Find the position in the second list for each element in the first list.

Uses:

Used in: `bcombine.m`, `bproject.m` and many other.

26 extstrmatch

Synopsis: EXTSTRMATCH finds the indices of a cell array of strings with respect to another cell array of strings.

Input:

- `srchforcellarray` is a cell array of strings to search for.
- `srchincellarray` is a cell array of strings in which search is done.

Output: `listindx` is a vector of indices.

Algorithm description:

- For each cell in searching array. A string match is carried.
- If the searching string is found in searched cell array then its index is recorded. Otherwise, index is set to 0.

Uses:

Used in: `uil2bm.m`

see also `xtfind.m+`

27 findneighbor.m

Synopsis: FINDNEIGHBOR finds the
neighbors of a given node in a binary join tree.

Input:

- `thenode` is a node id number for which neighborhood is looked for.
- `thebjtree` is bjtree id number in which neighborhood is searched.

Output: `neighbors` is the list of node id numbers.

Algorithm description:

- From the connection matrix, the list of neighbors is found.

Uses: `extfind`

Used in:

28 fosetrep.m

Synopsis: FOSETREP calculates the number of words and the size of last word.

Input:

- `cardivec` is the vector of cardinalities.
- `wordsize` is the number of bit of an integer used in representing bit vector.

Output: `repvec` is a vector of two elements `nwords` is the number of words and `sizelastword` is the size of the last word.

Algorithm description:

Calculate the size of join frame. Then determine the number words (each word is represented by an integer) needed and number of bits in the last word.

Uses:

Used in:

29 int2bitv.m

Synopsis: INT2BITV converts a integer matrix into bit matrix.

Input:

- `intmat` is the integer matrix to be converted.
- `rowlength` is the length of bit vectors
- `word_size` is the number of bits used from each integer except the last one in a row of integers. The default value is 50 bits.

Output: `bitvmat` is the bit matrix.

Algorithm description:

Calculate the size of last words.

Convert integers into bit strings.

Uses:

Used in:

see also `itmat2intmat.m+, \ver`+bvec2set.m+, `set2intmat.m`, `set2vec.m`

30 isbelequal.m

Synopsis: ISBELEQUAL
compares two lists of beliefs.

Input:

- `bellist1, bellist2` are two lists of belief id numbers
- `significance` is the tolerance allowed in comparison of two number.

Output:

- `comparesult = 1` if two input lists have the same length
  1. each corresponding pair of elements have the same domains (as sets but may have different order of variables)
  2. the same focal sets
  3. the same masses (masses are different by significance (default value is $1e-5$)
- Otherwise result is 0

Algorithm description:

Compare the domain sets.

Compare the number of foci.

Compare the mass vectors for each pair of beliefs.

Uses: `extfind, chngdomorder`

Used in:

31 keepbel.m

Synopsis: KEEPBEL
clear beliefs from memory except those in a given list

Input: `inbelidlist` is the list of id numbers for input beliefs.

Output: `belidlist` is the list of new id numbers for beliefs.

Algorithm description:

Clear all beliefs except those in the list.

Uses: `extfind`

Used in:

32 messagepass.m

Synopsis: MESSAGEPASS calculates the
message (belief potential) that is to be
passes from a node to another node in a binary join tree.

Input:

- `tree` is binary join tree number
- `fromnode` is a node id number from which message is going out
- `tonode` is a node id number to which message is coming to

Output: `messagebelid` is the belief id number representing the message.

Algorithm description:

Global BELIEF VARIABLE NODE BJTREE

Check consistency conditions.

Check if the required message already exists. If so do nothing.

Identify the neighbor nodes. Exclude the to-node to which a message is to be calculated. Calculate the message coming from each of remaining neighbors to the given node.

Combine the in-coming messages and project the combined belief on the intersection between domains of from-node and to-node.

Uses: `extfind, findneighbor, bproject, messagepass, multcombine`

Used in:

33 multcombine.m

Synopsis: MULTCOMBINE combines a list of beliefs.

Input: `list_bels`: the list of id numbers of beliefs to be combined.

Output: `combinedbel` the id number of the combined belief.

Algorithm description:

Global `BELIEF`

Check the validity of belief numbers.

Choose two beliefs in the list to do actual combination.

Update the list by

1. adding the newly created belief into the list and
2. remove two used beliefs from the list.

Uses: `extfind.m`, `varbelcross.m`, `choosetwobel.m`, `bcombine.m`, `belremove.m`, `beladd.m`.

Used in: `solve.m`

34 normalize.m

Synopsis: NORMALIZE does
normalization for the input beliefs. If the mass assigned
to the empty set is k then after normalization the empty set will be
removed from the set of foci and masses for other foci are multiplied by
factor 1/(1-k).

Input: `inbelidlist` is the vector of id belief numbers.

Output: `normbelidlist` is the vector of normalized beliefs.

Algorithm description:

Global	`BELIEF VARIABLE FRAME`
Check	consistency conditions.
Check	if there is a focus which is an empty set. If not, go to the next belief in the list.
Calculate	normalization factor.
Remove	the empty set focus.
Update	the masses for remaining foci.

Uses: `extfind`

Used in:

35 observe.m

Synopsis: OBSERVE creates a belief record that corresponds to an observation.

Input:

1. `varstr` is a string of variable names.
2. `instancestr` is a string of observations

Output: `belid` is the id number for observed belief.

Algorithm description:

Global `BELIEF VARIABLE FRAME`

Check consistency conditions.

Find id numbers of variables with names in the given string.

Create a belief domain with those id numbers.

Create a focus corresponding to the observation string.

Uses: `extfind, readwords, extstrmatch, bitmat2intmat`

Used in:

36 read1word.m

Synopsis: READ1WORD reads a word in a string beginning from a position.

Input:

1. `string` is a character string.
2. `position` is a number from which reading begins.

Output: `wordstruc` is a structure that has 2 fields.

1. `.word` is a char string.
2. `.lastpos` is a integer indicating the last position of the word.

Algorithm description:

Move from the beginning position and skip space and delimiter symbols which are `{},()`.

Read characters of the string until a delimiter symbol is met.

Uses:

Used in:

37 readwords.m

Synopsis: READWORDS reads words from a char string.

Input: `strvec` is a character string.

Output: `cellwords` is a cell array of words.

Algorithm description:

Read consecutively words from the string.

Uses: read1word

Used in:

38 removeavar.m

Synopsis: REMOVEAVAR removes a variable in from the pool of variables to be processed in the
fusion algorithm.

Input:

- `invarbelstruct` is the variable-belief structure containing 5 fields
  1. `varbelstruct.varnums`: the list of variable id numbers
  2. `varbelstruct.varcard`: the list of variable cardinalities
  3. `varbelstruct.belnums`: the list of belief id numbers
  4. `varbelstruct.belcard`: the list of belief cardinalities (numbers of focal sets)
  5. `varbelstruct.cross`: the table of cross reference
- `vartoremove`: the variable to be deleted.

Output: `varbelstruct` is the updated variable-belief structure.

Algorithm description:

Global	`BELIEF`
Identify	the beliefs that has the variable in their domains.
Combine	those beliefs by using `multicombine.m`.
Project	the combined belief on the domain with the variable deleted.
Update	variable-belief structure.

Uses: `var2bel.m`, `multcombine.m`, `bproject.m`, `belremove.m`, `beladd.m`.

Used in: `solve.m`

39 roundedmat.m

Synopsis: ROUNDING rounds numbers in a matrix to certain significant digits.

Input:

- `mat` is a matrix.
- `sigdigits` is a integer number.

Output: `varbelstruct` is the updated variable-belief structure.

Algorithm description:

Rounding number in the matrix to certain significant digits.

Uses:

Used in:

40 senanalysis.m

Synopsis: SENANALYSIS does
sensitivity analysis on the effect of variation of mass
assignment on one belief on another marginal.

Input:

1. `belinput` is belief input whose mass assignment will be varied
2. `massvecmat` is the matrix of mass vectors
3. `targetvar` is the set of target variables
4. `userbellist` the list of belief id numbers that constitute the belief base

Output: `beloutid` is the id number of output belief.

Algorithm description:

Global	BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME
Check	consistency conditions
Replace	mass vector by the input matrix of mass vectors.
Calculate	the marginal for the required set of variables.

Uses: `extfind, readwords, extstrmatch, solve`

Used in:

41 set2intmat.m

Synopsis: SET2INTMAT converts the set representation of foci into integer representation.

Input:

- `vars`: the list of variable id numbers in the domain of belief whose focal sets are given.
- `belstruct` is a structure containing 2 components
  1. `belstruct.domain`: the list of variable id numbers in the belief
  2. `belstruct.indices`: a 2-D integer array. Each row is coordinates one element of a focal set. Each column corresponds to a variable. There are special conventions as follows
     - a row consists of -1s tells the end of a focal set,
     - an index that equals the $c+1$ where c is the cardinality of the variable is a wildcard that can substitute for any value. For example if cardinalities of A and B are 60 and 2, a row [61 3] denotes the whole frame AB.
     - a row of 0s denotes the empty focal set.

Output: `intfocals` is the obtained array where each row is an integer representation of a focal set. Each integer in a row is converted from 50 bits, except the last integer.

Algorithm description:

Check the validity of input.

Check the presence of a empty focal set.

Handle the wild cards.

Convert set representation into integer presentation.

Uses:

Used in: `anexample.m`

see also `et2vec.m+`

42 showbel.m

Synopsis: SHOWBEL shows a list of beliefs in a table form. It shows a one belief after another.

Input:

- `toshowlist` is the list of belief id numbers to be shown.
- `showform` is a character string that indicates either an element is shown as code/index (more compact) or label (readable). The label form is default. To show in code form `showform = 'code'`.

Output: is a print-out of a table. The first row contains the variable names. Each focal set is a segment bordered by 2 lines of separation. Each row is the vector of coordinate of an element of the focal set. The last column indicates masses. The empty focal set is represented by a blank row. The whole frame focal set is represented by a string of asterisks (*).

Algorithm description:

Global	`BELIEF VARIABLE FRAME`
Check	the validity of the id numbers of beliefs.
Change	the form of focal sets from integer vector to set of elements.
Retrieve	variable information and merge into one table. Then print it.

Uses: `extfind.m`, `varreference.m`, `int2bitv`, `bvec2set`, `code2label`.

Used in:

43 showtree.m

Synopsis: SHOWTREE shows structure of the binary join tree (non-graphic).

Input: `treeidnumber` is the tree id number.

Output: 2 matrices

1. 1st matrix consists of 4 columns: row-node, col-node, row-col message,

   col-row message; number of rows is the number of edges

2. 2nd matrix on nodes. It has 3 columns: node id, variables in node,

   potential

Algorithm description:

Global `BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME`

`QUERY BELTRACE NODE BJTREE`

Check consistency conditions.

Print informations on two matrices.

Uses:

Used in:

44 solve.m

Synopsis: SOLVE

calculate the marginal belief on the list of variables from the beliefs given in a list.

Input:

- `list_bels` is a list of beliefs relevant to the calculation.
- `list_vars` is a list of variables.
- `normmode` normalization mode. default 0: no normalization; 1: normalization
- `threshold` approximation threshold. default 0: no approximation; 1: approximation with default threshold; x: approximation with threshold x.

Output: `bel_idnum` is the id number of the computed marginal belief.

Algorithm description:

Global `BELIEF QUERY`

Build the variable-belief structure for those beliefs given in the input.

Determine the join domain and the set of variables to be deleted according to the fusion algorithm.

Select a deletion sequence of variables.

Remove one by one variable in the sequence until only variables for which marginal is required remain. Combine the remaining beliefs.

Create a record of query.

Uses: `varbelcross.m`, `chooseaseq.m`, `removeavar.m`, `multcombine.m`

Used in: `ssolve.m`

see also olve.m+

45 solvetreeall.m

Synopsis: SOLVETREELL
find marginals for each individual variable based on a tree.

Input: `tree` is a binary join tree id number.

Output: `beloutlist` is the vector of belief id numbers of required marginals.

Algorithm description:

Global `BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME QUERY`

`BELTRACE NODE BJTREE`

Check consistency conditions.

Identify the list of nodes such that the union of their labels is the set of all variables.

Calculate marginal for each of node in the list.

Project from the node marginals on each individual variable.

Uses: `extfind, solvetreenode, bproject`

Used in:

see also `olve.m+`

46 solvetreenode.m

Synopsis: SOLVENODE
finds marginal of a node in a binary join tree.

Input:

1. `tree` the id number of a tree
2. `finalnode` the id number of a node for which marginal is sought.

Output: `beloutid` is the id number of a calculated belief function.

Algorithm description:

Global `BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME`

`QUERY BELTRACE NODE BJTREE`

Check consistency conditions.

Start from the node for which marginal is required.

Issue recursively request for all in-coming messages from its neighbors.

Combine all in-coming messages with the potential present at the node.

Uses: `extfind, findneighbor, messagepass, multcombine`

Used in:

see also `olve.m+`

47 solvetreevar.m

Synopsis: SOLVETREEVAR
find a marginal for the list of variables based on the structure of a tree.

Input:

1. `tree` is the id number of a tree.
2. `varlist` is list of variables for which a marginal is sought.

Output: `belout` is the id number of a calculated belief function.

Algorithm description:

Global `BELIEF VARIABLE ATTRIBUTE STRUCTURE FRAME QUERY BELTRACE NODE BJTREE`

Check consistency conditions.

Identify the node whose label includes the set of variables for which a marginal is required. If it exists.

Call `solvetreenode` to find the marginal and then project on the set of variables.

Call `solve` to directly find the marginal without using the tree.

Uses: `extfind, readwords, extstrmatch, solve, solvetreenode, bproject`

Used in:

see also `olve.m+`

48 ssolve.m

Synopsis: SSOLVE

calculate the marginal belief on the list of variables from the beliefs given in a list. The same function that SOLVE performs. But in distinction to SOLVE, SSOLVE uses the information contained in QUERY database to see how much computation can be reused.

Input:

- `list_bels` is a list of beliefs relevant to the calculation.
- `list_vars` is a list of variables.

Output: `bel_idnum` is the id number of the computed marginal belief.

Algorithm description:

Global `BELIEF QUERY`

Determine the variable-belief structure for relevant beliefs.

Find the set of variables to be deleted. Select a deletion sequence.

Select from `QUERY` those records that have the same belief input as the current query.

Find a largest deletion subsequence that already done (from QUERY structure)

Update the list of beliefs and create a new query (call to function `solve.m`

Uses: `varbelcross.m`, `local_selectquery.m`, `chooseaseq.m`, `var2bel.m`, `solve.m`.

Used in:

see also `olve.m+`

49 uil2bm.m

Synopsis: UIL2BM

converts belief data declared in Unified Interface Language (UIL) to the format that BFM accepts.

Input:

- `uilfilename` is the full name (default extension '.txt') of the text file that contains the data in UIL format.
- `bmfilename` is the name for the MATLAB file (extension .m) that contains converted BFM data.

Output: `nrofbelief` the number of beliefs in the converted data.

Algorithm description:

Global `VARIABLE FRAME BELIEF`.

Read the text file and convert it into cell array of statements. This is done by function `local_readstmt` that removes commentaries, multiple spaces and puts data belonging to one statement but spreads on multiple lines into a statement cell.

Process statement one by one

DEFINE VARIABLE. Information given this statement channels into 2 structures: `bmframe` and `bmvariable` (corresponding to BFM's VARIABLE and FRAME).

DEFINE RELATION. Information given in this statement is kept temporarily in structure `uilrelations` which does not have counter part in BFM format.

SET VALUATION or COMBINE VALUATION. Information given in these statements channels into `fosets` matrices that contains focal sets in the set form and `massvalues` vector that contains masses assigned to focal sets.

Convert focal sets given in the set form into the integer vector form used in BFM's BELIEF structure.

Uses: `extstrmatch`, `local_cleanup`, `local_readwords`, `local_r1word`

Used in:

49.1 local_readstmt

Synopsis: LOCAL_READSTMT reads lines from a text file and put statements in an cell array.

Input: `readfname` is a text file name.

Output: `stmtcellarray` is an array of cells in each them is a statement in UIL.

Algorithm description:

Constants are set. The maximum number of statements is set to 1000, the maximum length of a statement to 5000 characters.

Lines of the text file are read one by one. If the line is empty or is a comment line, it is ignored.

Clean up the line by function `local_cleanup`. This function among other things removes unnecessary blanks.

Process a line. If a statement is contained within a line, it is put immediately into a cell. Otherwise, data in a line is put into a buffer `tmp_stmt` until a statement is complete.

Uses: `local_cleanup`

Used in: `uil2bm.m`

49.2 local_cleanup

Synopsis: LOCAL_CLEANUP cleans a text line from comments and unnecessary blanks.

Input: `inline` is a line of text.

Output: `outline` is cleaned line.

Algorithm description:

Remove all characters following a comment symbol.

Remove unnecessary multiple blanks and tab symbols.

Uses:

Used in: `local_readstmt`

50 var2bel.m

Synopsis: VAR2BEL finds the list of belief id numners that include a variable given in a list in its domain.

Input:

- `varbelstruct` is a variable-belief structure that has the following fields
  1. `varbelstruct.varnums`: the list of variable id numbers
  2. `varbelstruct.varcard`: the list of variable cardinalities
  3. `varbelstruct.belnums`: the list of belief id numbers
  4. `varbelstruct.belcard`: the list of belief cardinalities (numbers of focal sets)
  5. `varbelstruct.cross`: the table of cross reference
- `var_list` is the list of variable id numbers.

Output: `belinvolved` is the list of belief id numbers involved.

Algorithm description:

Find the indices (positions) of the given variables in the variable-belief structure.

Determine the list of beliefs involved.

Uses: `extfind.m`

Used in: `solve.m`

see also `el2var.m+`

51 varbelcross.m

Synopsis: VARBELCROSS summaries the relationship between variables and beliefs for those contained in a list of belief id numbers.

Input: `belief_list` is the list of belief id numbers.

Output: `varbelstruct` is a variable-belief structure that has the following fields

1. `varbelstruct.varnums`: the list of variable id numbers
2. `varbelstruct.varcard`: the list of variable cardinalities
3. `varbelstruct.belnums`: the list of belief id numbers
4. `varbelstruct.belcard`: the list of belief cardinalities (numbers of focal sets)
5. `varbelstruct.cross`: the table of cross reference such that

   `valbelstruct.cross(i,j)=1` to means that `varnums(i)` is present in the domain of `belnums(j)`

Algorithm description:

Global `BELIEF VARIABLE`

Retrieve information about beliefs given in the list from `BELIEF` structure. From that determine the set of variables involved.

Retrieve information about the variables by module `varreference.m`

Create variable-belief structure.

Uses: `extfind.m`, `varreference.m`

Used in: `solve.m`, `ssolve.m` and others.

52 varreference.m

Synopsis: VARREFERENCE creates a reference structure for variables given in a list.

Input: `var_list` is the list of variable id numbers.

Output: `varnumcard` is a structure of 2 fields

1. `varnumcard.nums`: the sorted list of variables.
2. `varnumcard.card`: the list of their cardinalities.

Algorithm description:

Global `VARIABLE FRAME`

Find the cardinality of variable's frame for each of the input list.

Uses: `extfind.m`

Used in: `varbelcross.m`