Laurea Specialistica in Informatica
a.a. 2005-2006

Interazione Uomo-Macchina II:

Tool per Belief Networks*

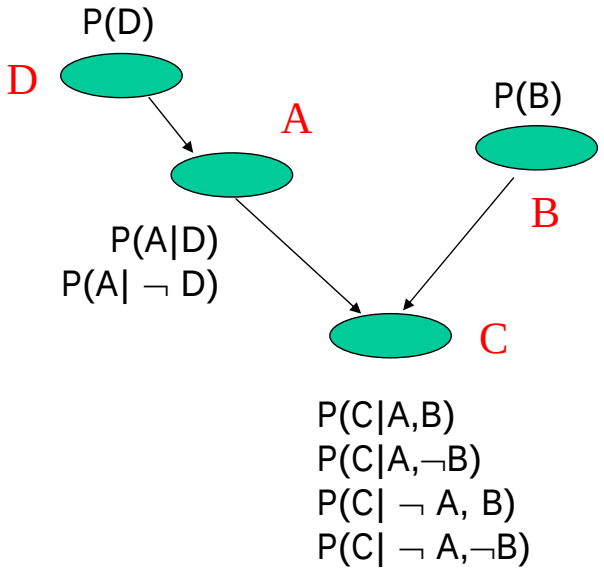
Irene Mazzotta
mazzotta_AT_di.uniba.it

Ringrazio
Giovanni Cozzolongo e Vincenzo Silvestri per il contributo nella realizzazione delle slides.

Belief Network (o Bayesian Network, o Reti Causali Probabilistiche (RCP)):

- *Bayesian network* (*BN*) è un formalismo per modellare un dominio contenente varie forme di incertezza.
- A BN è un grafo orientato, aciclico, i cui nodi rappresentano variabili a più valori e gli archi rappresentano la *relazione causale* fra i nodi che collegano.
- La forza di queste relazioni è misurata in termini di *probabilità condizionate*.
- Ad ogni nodo radice (che non ha genitori), è associata una *tabella di probabilità marginale.* A tutti gli altri nodi è associata una *tabella di probabilità condizionata*

Un esempio semplice di RCP

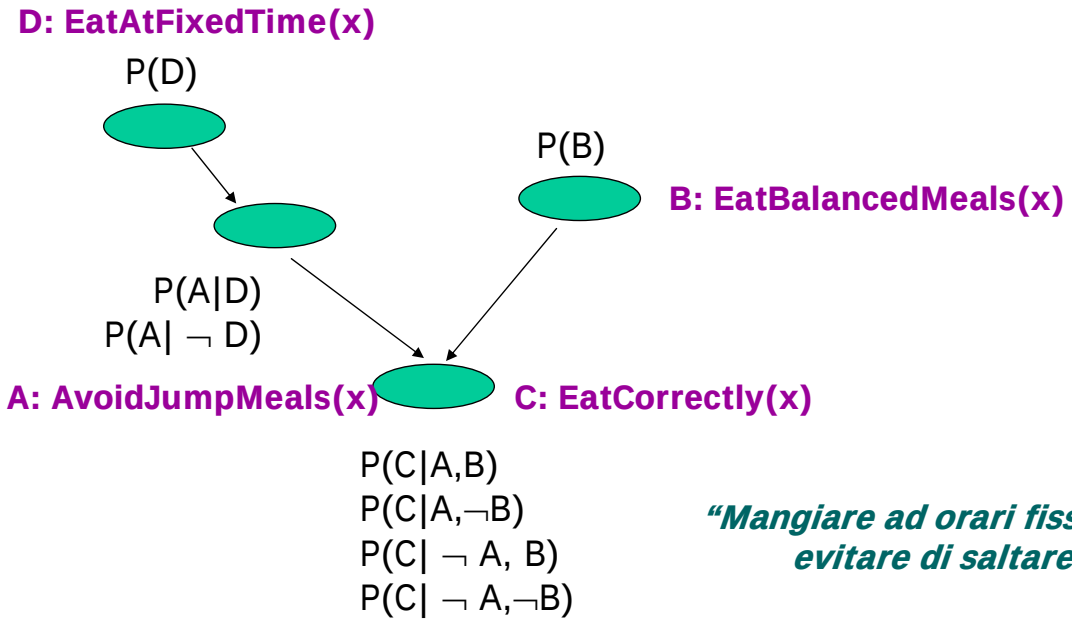


Il grafo rappresenta ipotesi di 'indipendenza condizionale' tra le variabili associate ai suoi nodi:

A è indipendente da B e da C, condizionatamente a D:
P(A|B,C,D) = P(A|D)

C è indipendente da D, condizionatamente ad A:
P(C|A,B,D) = P(C|A,B)

Cosa potrebbe rappresentare la RCP che abbiamo appena visto



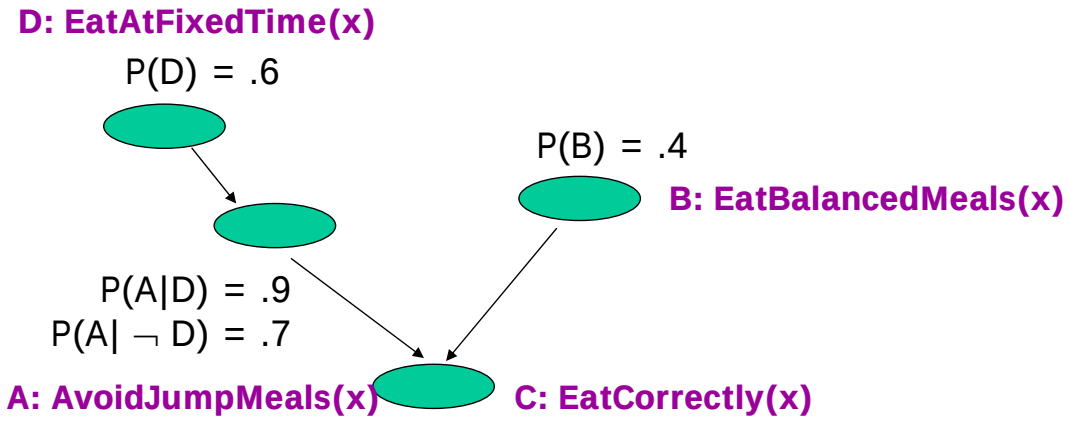
"Mangiare ad orari fissi aiuta ad evitare di saltare i pasti ..."

"Evitare di saltare i pasti e mangiare pasti bilanciati contribuiscono a mangiare correttamente"

Fasi nella Costruzione di una RCP

1. Stabilire la differenza fra variabili 'nascoste' (hidden) e 'osservabili'
2. Definire la struttura del grafo:
   - semantica dei nodi
   - semantica delle relazioni
   - attenzione alle ipotesi di indipendenza condizionale!
3. Assegnare i parametri
4. Propagare nuova evidenza nella rete

… Continuiamo l'esempio



Prova a calcolare
P(C) e P(A)

Tool per BN

Le operazioni che un tool deve permettere sono:

- *creare, aggiornare, ecc 'grafi orientati'*
- ai cui nodi-radice e ai cui archi sono associate delle misure di 'incertezza',
- *propagare 'evidenza'* nota su alcuni nodi,
- osservare come varia l'incertezza associata agli altri nodi.

Ci sono diversi tool per fare questo:

Hugin, eBayes/javaBayes, Netica, Analytica, ecc...

Tool per BN

Hugin:
sw commerciale per Belief Network.
Fornisce API per C, C++, Java.

eBayes/javaBayes
sw open source completamente scritto in java

Netica
sw commerciale con API per C, Java e VB.

Analytica
sw commerciale per la creazione, analisi e comunicazione di decisioni con BN e Influence Diagram.

Genie, MSBN, ecc…

Hugin components

Hugin Development Environment ha tre componenti:

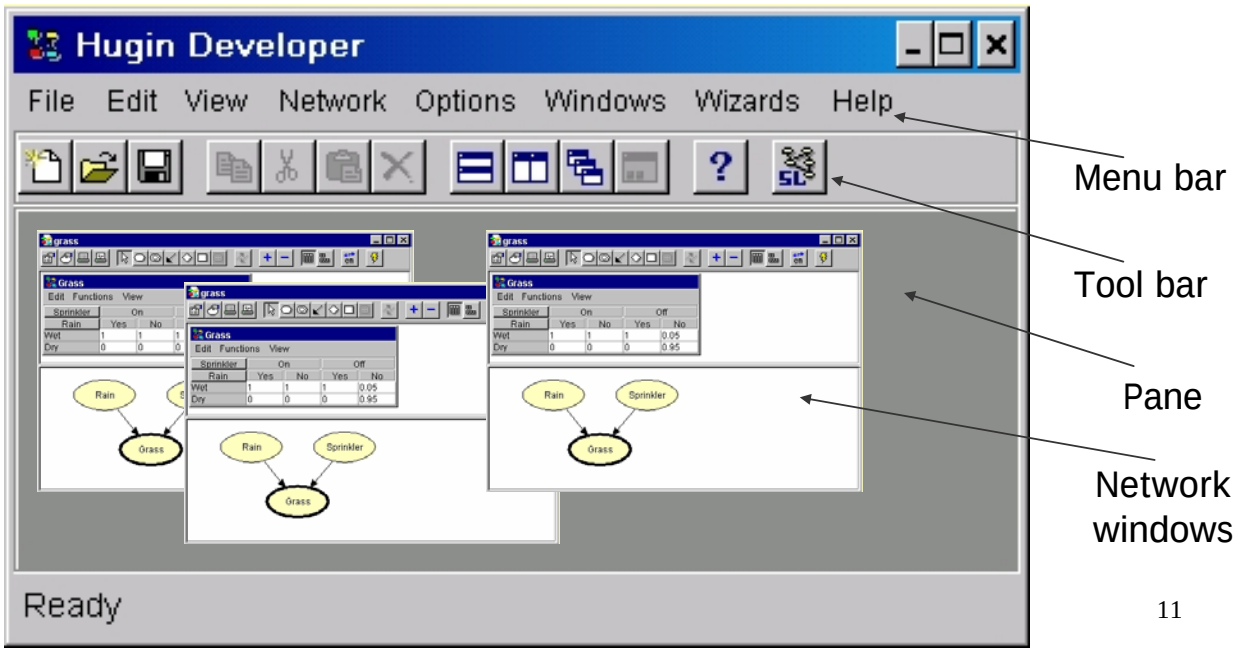
- Hugin Decision Engine (HDE): è il motore inferenziale di Hugin; esegue il ragionamento sulla KB rappresentata mediante Bayesian Network; una parte importante dell'HDE è il *compiler* che trasforma le reti in strutture funzionali (junction trees), rendendo possibile le inferenze (reasoning) in the network.
- Hugin fornisce due possibili accessi all'HDE: *Hugin Graphical User Interface* and *Hugin Application Program Interfaces*.

Hugin Graphical User Interface

- *Hugin Graphical User Interface* è usato per creare, memorizzare ed eseguire (inserendo e propagando evidenze) modelli di network.
- Consiste di due modalità operative: *Edit Mode* and *Run Mode*.
- La modalità *edit* è usata per creare i nodi e link fra questi, gli stati dei singoli nodi, le tabelle di probabilità condizionata e marginale. Tutte queste operazioni vengono eseguite mediante *a window-, menu- and mouse driven interface*.
- Nella modalità *Run* l'utente può inserire evidenze sui nodi osservabili attraverso la selezione degli stati interessati. Hugin Decision Engine provvederà a propagare le informazioni inserite e a rivedere le probabilità dei nodi coinvolti.
- una collezione di Application Program Interfaces (API), e un Hugin Graphical User Interface .

Hugin Graphical User Interface: Main Window

Contiene le *network windows* ognuna delle quali visualizza una Hugin network.



Main Toolbar di Hugin

Toolbar di Bottoni:

- **File** : operazioni sul *file-network* (new, open, save, save as, …)
- **Edit**: operazioni su *parti del network* (cut, copy, … applicate a *nodi, loro stati, link, tabelle di probabilità associate ai link*, ….)
- **View**: operazioni sulle finestre di lavoro successive
- **Network**: funzioni di *propagazione dell'incertezza* nel network
- **Options** : operazioni sulla *finestra di lavoro*
- **Window**: "
- **Wizard**: procedure interattive per l'apprendimento di reti da basi di dati
- **Help**: ?

Main Toolbar di Hugin

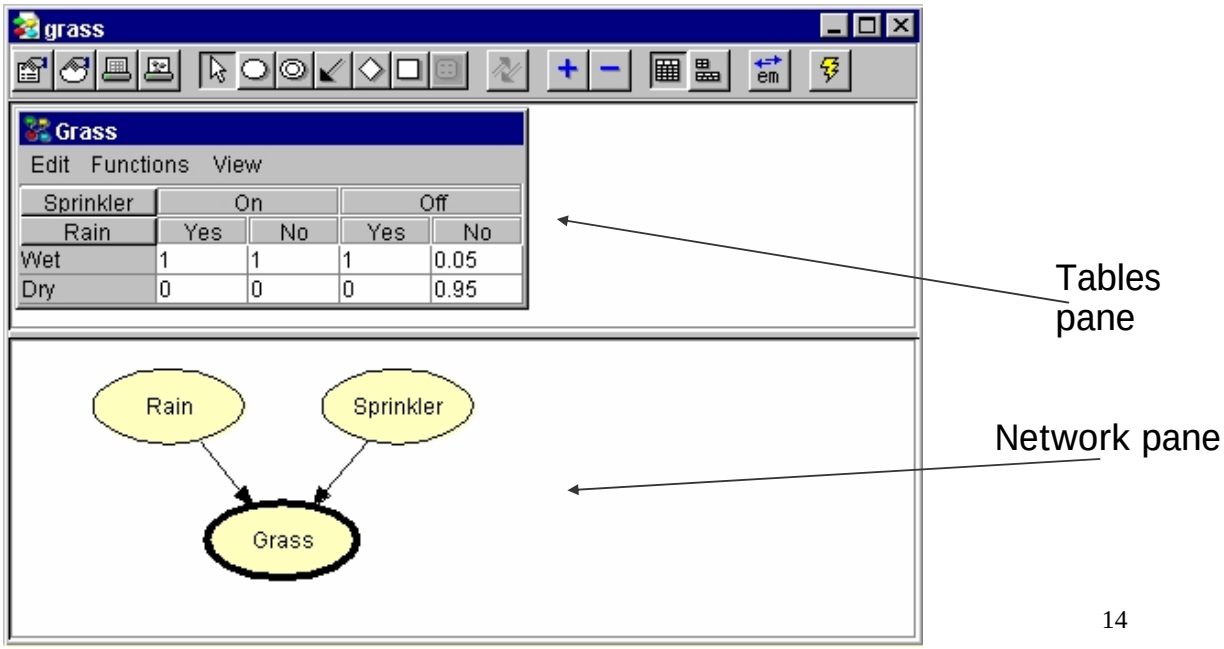
Toolbar di Icone:

- riproduce, in forma di icone, alcune delle operazioni nel file di bottoni (quelle ritenute più 'frequenti');
- le organizza in 'gruppi di icone', sulla base del bottone a cui fanno riferimento:
    - gruppo 1: operazioni sui file (new, open, save, print network)
    - gruppo 2: operazioni su parti del network (cut, copy, paste, delete)
    - gruppo 3: funzioni di propagazione dell'incertezza, …. Ecc

  e aggiunge una icona di 'help'

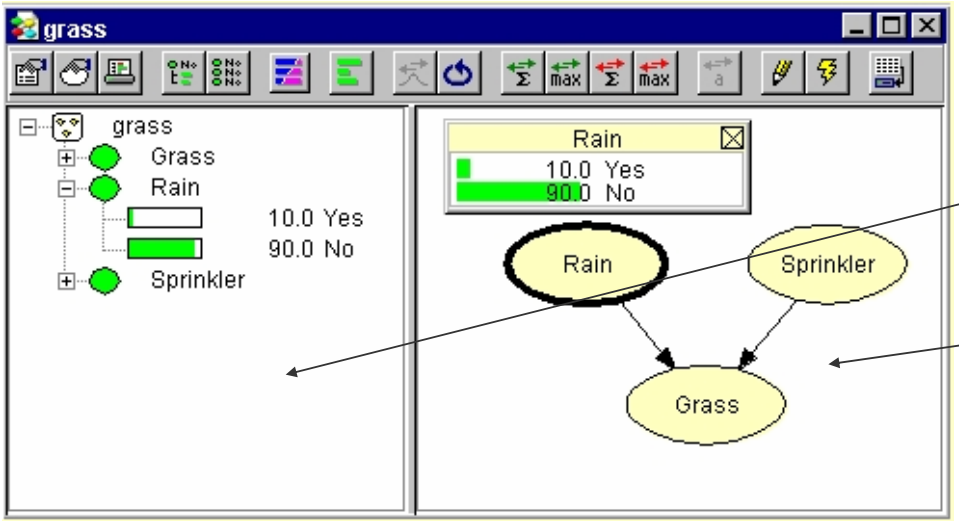
Hugin Graphical User Interface: Edit mode

Per costruire una Hugin networks the Hugin Graphical User Interface deve lavorare in *Edit mode*.



Hugin Graphical User Interface: Run mode

Per usare una Hugin networks the Hugin Graphical User Interface must be working in *Run mode*.



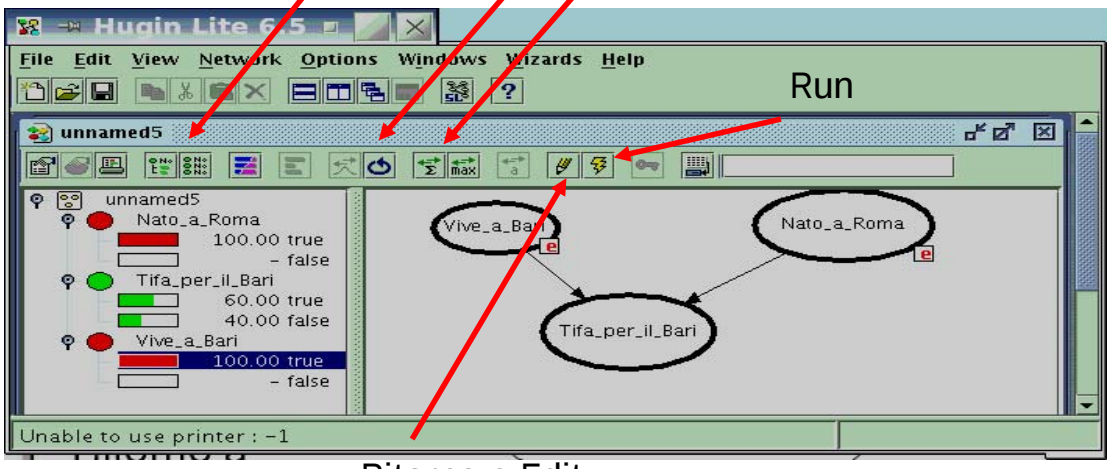
Finestra di 'Run'

Anche qui, toolbar organizzata per gruppi di bottoni

I quattro che seguono, di visualizzazione del frame di sinistra

Reset

Metodi di propagazione dell'evidenza



Ritorno a Edit

Oggetti e Funzioni in Hugin

Oggetti:

- La *finestra di lavoro*
- Il *file-network*
- Il *network*, con le sue parti: *nodi e link fra nodi*
- Le *proprietà di nodi e link*: nomi, tipo, stati, ...
- L'*incertezza: probabilità a priori, probabilità condizionate, evidenza sui nodi, ...*

Oggetti e Funzioni in Hugin

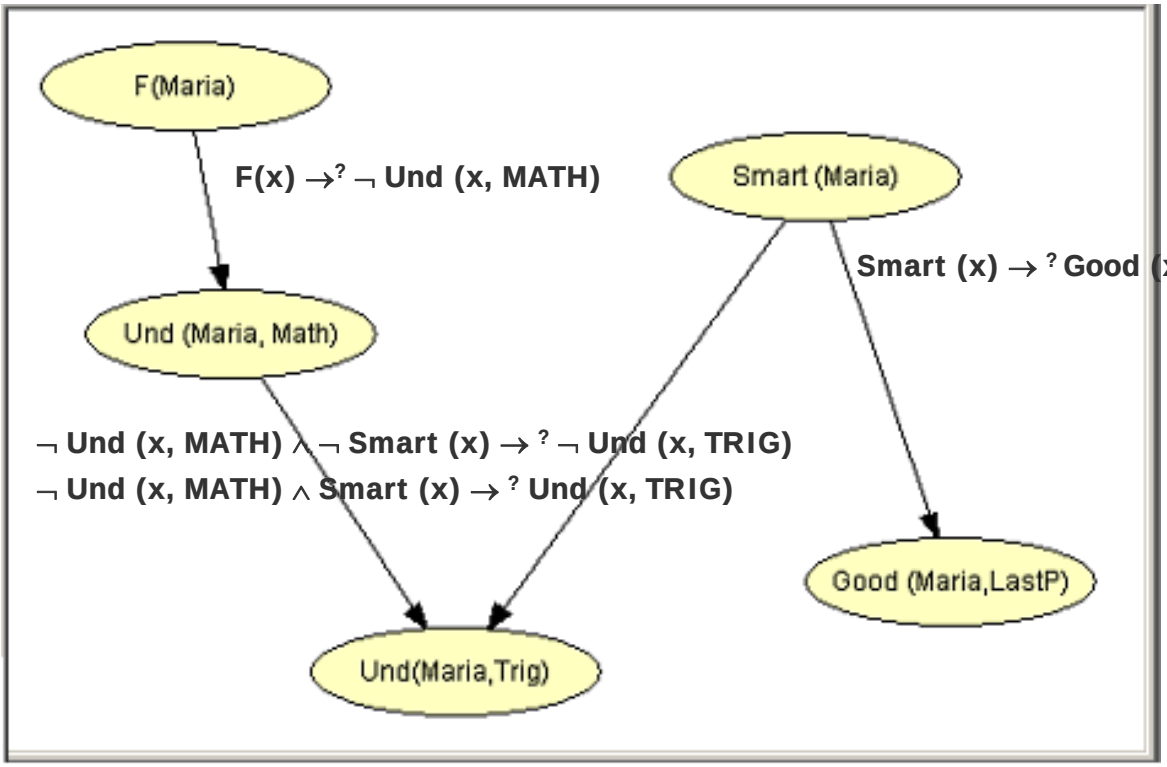
Funzioni:

- visualizza, non visualizzare (per gli oggetti nelle varie finestre di lavoro)
- crea, apri, salva, stampa, chiudi (per i file-network)
- cancella, taglia, copia, aggiungi, modifica parti del network (edit)
- inizializza il network propagando i parametri inseriti (run)
- propaga l'evidenza sui nodi
- undo l'evidenza propagata
- visualizza gli effetti della propagazione dell'evidenza,
- .....

Esempio : base di conoscenza
"*come formalizzare un pregiudizio...;-)*"

Le ragazze non capiscono, in genere, la matematica.
Chi non capisce la matematica e non è particolarmente intelligente ha, in genere, difficoltà a capire la trigonometria.
Chi è intelligente ha dato, in genere, prove precedenti di bravura.

Formalizziamo la Conoscenza Generale



Inseriamo i Parametri nel BN

P(F(x)) = .5 (nella classe, c'è la stessa proporzione di ragazze e di ragazzi)

P(Smart(x))= .5 (nella classe, c'è la stessa proporzione di studenti intelligenti e non intelligenti)

P(Good(x, LASTP)|Smart(x)) = .9 (se uno studente è intelligente, molto probabilmente ha dato prove precedenti di bravura)

P(Good(x, LASTP)|¬Smart(x)) = .4 (se uno studente non è intelligente, è poco probabile che abbia dato prove precedenti di bravura)

P(Und(x, Math)|F(x)) = .3 (una ragazza, è poco probabile che capisca la matematica)

P(Und(x, Math)|¬F(x)) = .6 (un ragazzo, è più probabile che capisca la matematica)

Inseriamo i Parmetri nel BN (segue)

P(Und(x, Trig)|Und(x, Math), Smart(x)) = .95 (se uno studente capisce la matematica ed è intelligente, quasi certamente capisce anche la trigonometria)

P(Und(x, Trig)| ¬Und(x, Math), Smart(x)) = .5 (se uno studente non capisce la matematica ma è intelligente, non si può prevedere se capisce o no la trigonometria)

P(Und(x, Trig)|Und(x, Math), ¬Smart(x)) = .6 (se uno studente capisce la matematica ma non è intelligente, potrebbe anche capire la trigonometria)

P(Und(x, Trig)| ¬Und(x, Math), ¬Smart(x)) = .1 (gli studenti che non capiscono la matematica e non sono intelligenti, molto probabilmente non capiscono neanche la trigonometria)

Divertiamoci con Hugin

Formalizziamo mediante Hugin Belief Network ciò che segue:

Le ragazze non capiscono, in genere, la matematica.
Chi non capisce la matematica e non è particolarmente intelligente ha, in genere, difficoltà a capire la trigonometria.
Chi è intelligente ha dato, in genere, prove precedenti di bravura.

esempio1.net

"Mangiare ad orari fissi aiuta ad evitare di saltare i pasti …"
"Evitare di saltare i pasti e mangiare pasti bilanciati contribuiscono a
mangiare correttamente"

esempio2.net

Hugin APIs

- The *Hugin APIs* (Application Program Interfaces): per costruire applicazioni knowledge-based, che sfruttano la potenza dell'HDE quale motere inferenziale.
- Forniscono i metodi per inserire il motore inferenziale di Hugin all'interno di un'applicazione che abbia come base di conoscenza dei BNs
- Hugin APIs sono disponibili come C, C++, e Java libraries e come ActiveX server.
- Una breve lista delle più importanti funzioni fornite dall'HDE attraverso le APIs è disponibile alla pagina Feature Lists del web site di Hugin. Inoltre, Hugin API reference manuals possono essere scaricate dal web site di Hugin.

Api JAVA di Hugin

- La libreria delle HUGIN API si trovano in due file:
  - hapi63.jar : l'interfaccia Java alla libreria sottostante in C. deve trovarsi nel **CLASSPATH**
  - libhapi63.dll : la libreria che deve trovarsi in una sottodirectory dell'applicazione o come argomento della VM con l'opzione **-Djava. library.path**

- La classe che dovra' gestire l'interazione con le API dovra' contenere l'import relativo
  - import COM.hugin.HAPI.*;

Api di Hugin: le classi base

- Domini, nodi ecc sono modellati come classi.
- I metodi della classe ***domain*** permettono di gestire una rete (creare un dominio vuoto/da file, compilarlo, salvarlo, propagare le evidenze, ecc.)
- I metodi della classe ***node*** permettono di gestire I nodi all'interno della rete (aggiungere, togliere, modificare attributi, settare le evidenze, attraversare il grafo, ecc.)

Api di Hugin: creare un dominio

- Per inserire i nodi e' necessario definire un dominio;
- Un dominio contiene il belief network che verra' elaborato. Questo puo' essere creato da codice o caricato da un file:
  - Es1: dominio1 = new Domain(); // crea un dominio vuoto
  - Es2: dominio2 = new Domain("nomefile.hkb") // carica un dominio da file
- La creazione di un dominio puo' essere inserita in un try – catch per verificare la corretta inizializzazione e individuare gli errori.

Api di Hugin: esempio

```
try {
dominio = new Domain("rete.hkb");
   }

   catch (ExceptionHugin EH) {
     System.out.println(EH.getMessage());
     System.out.println("errore1");
     EH.printStackTrace(System.out);
     System.exit(1);
   }
```

Api di Hugin: I nodi

- Sia che il dominio sia vuoto o che lo si carichi da file e' necessario definire degli oggetti della classe **node** che conterrano I loro valori. I nodi possono essere di tipi diversi:
  - BooleanDCNode, (true-false)
  - IntervalDCNode, (0-1,1-2, ecc.)
  - LabelledDCNode, (etichettati)
  - NumberedDCNode (etichettati con un numero)

Es:... LabelledDCNode nodoEtichettatoN;
BooleanDCNode nodoBooleanoM; ...

Api di Hugin: leggere I nodi

- Agli oggetti così creati vanno associati I valori del dominio caricato dal file nel seguente modo:

```
try {…
nodoEtichettatoN =
  (LabelledDCNode)dominio.getNodeByName("EtichettaN");
nodoBooleanoM =
  (BooleanDCNode)dominio.getNodeByName("BooleanM");
      catch (ExceptionHugin ex) {
  System.out.println(ex.getMessage());
  System.out.println("errore2");
  ex.printStackTrace(System.out); System.exit(1);
      }
```

Api di Hugin: settare I nodi

- Se si aggiunge un nodo al dominio bisogna settare le informazioni del nodo.
  - Impostare il numero di stati:
    - setNumberOfStates (int newNumber)
  - Impostare l'etichetta di un nodo:
    - setStateLabel(int state, java.lang.String newLabel)
  - Definire I nodi genitori del nodo:
    - addParent (Node nomeNodoGenitore)

- Per eliminare un nodo si usa il metodo **delete()**

Api di Hugin: metodi utili (1)

- E' possibile accedere a tutte le informazioni su un nodo:

- NodeList getChildren() restituisce una lista dei nodi figli
- NetworkModel **getHome**() restituisce la classe o il dominio a cui appartiene il nodo
- Class **getHomeClass**() restituisce la classe che contiene il nodo
- Domain **getHomeDomain**() restituisce il dominio che contiene il nodo

Api di Hugin: metodi utili (2)

- NetworkModel.Kind **getKind**() restituisce il tipo di nodo (discreto, continuo, ecc)
- String **getLabel**() restituisce l'etichetta del nodo
- String **getName**() restituisce il nome del nodo
- NodeList **getParents**() restituisce una NodeList con i nodi genitori
- java.awt.geom.Point2D **getPosition**() Restituisce la posizione del nodo

Api di Hugin: metodi utili (3)

- Si puo' controllare lo stato di un nodo :
- boolean **evidenceIsEntered**()
  - Vero se e' stata imposta un evidenza sul nodo
- boolean **evidenceIsPropagated**()
  - vero se l'evidenza e' gia' stata propagata
- boolean **evidenceToPropagate**()
  - vero se l'evidenza imposta deve essere ancora propagata.
- Table **getTable**()
  - restituisce la tabella delle probabilita' del nodo

Api di Hugin: metodi utili (4)

- Per ottenere il calcolo della propagazione di evidenze nella rete usiamo:

- Node.selectState(state) setta l'evidenza sullo stato *state*

- dominio.compile(); compila il dominio con le evidenze impostate

- dominio.propagate(dominio.H_EQUILIBRIUM_SUM, dominio.H_EVIDENCE_MODE_NORMAL); propaga le evidenze.

Api di Hugin: stampa di BN

- Per stampare i risultati della propagazione e' necessario scandire tutto il grafo e ottenere i belief di tutti gli stati di tutti i nodi:

- double **getBelief**(int state) restituisce il valore dello stato *state*

- Bisogna costruire un ciclo partendo da un nodo analizza tutti gli altri

Api di Hugin: esempio stampa di BN

```
public void StampaBN(Domain domain) {
try {
      Node node;
      ListIterator it = domain.getNodes().listIterator();
      String vettore[];
      String indice1[];
      ...
node = (Node)it.next();
while(it.hasNext()) {
...
If (( node.getName().equals("nomeNodo1"))| …) {
...
}
for (x=0;x<=lunghezzaVettore;x++) {
      System.out.println(indice1[x]+": "+ vettore[x]);
      }
 } catch (ExceptionHugin e) {
 System.out.println("Exception caught:");
 System.out.println(e.getMessage());
} } }
```


Api di Hugin: esempio stampa di BN

```
If (( node.getName().equals("nomeNodo1"))|          …          |
                    … |( node.getName().equals("nomeNodoN")) {

  if (node.getKind() == Domain.H_KIND_DISCRETE) {
        indice1[x]=node.getName();
        for (int i = 0; i < ((DiscreteChanceNode)node).getNumberOfStates(); i++) {
              if (max_stato<((DiscreteChanceNode)node).getBelief(i)) {
                    max_stato=((DiscreteChanceNode)node).getBelief(i);
                    stringa= ((DiscreteChanceNode)node).getStateLabel(i);
              vettore[x]=stringa;
              }
        }
        x++;
      }
}
```

Belief Network Dinamici

- Metodo per rappresentare conoscenza in domini "inerentemente incerti"
- Natura statica delle applicazioni: *ogni variabile è osservabile una sola volta*.

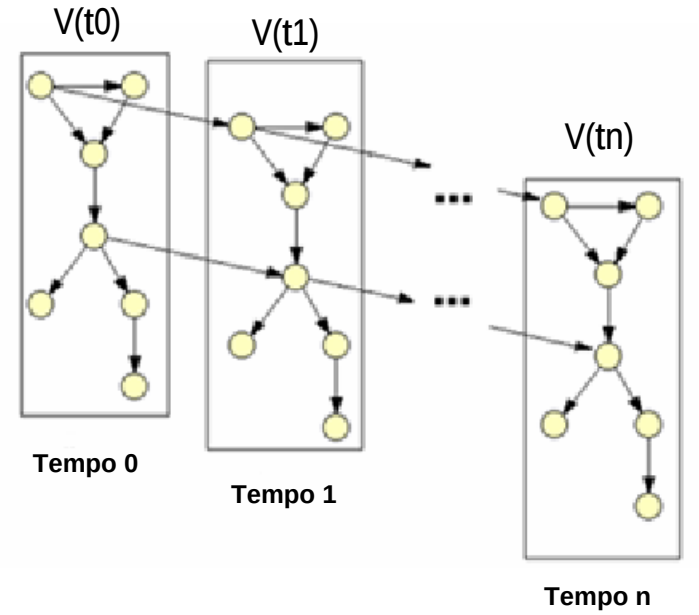
Belief Network Statici

- Metodo per rappresentare domini che comportano anche *osservazioni ripetute di variabili* random
- Comporta la connessione fra istanze multiple di belief network statici

Belief Networks Dinamici

DBN: una definizione generale…

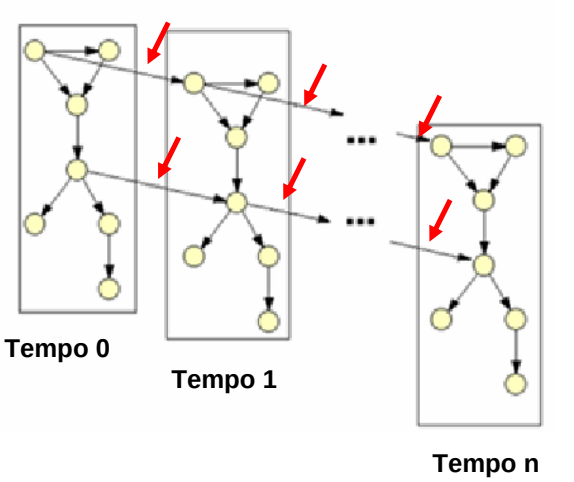
Un modello dinamico è una sequenza di sottomodelli, ciascuno dei quali rappresenta lo stato del sistema in un certo istante di tempo



DBN = {V,E}, dove:
Se
• $V(t_i)$ descrive la struttura del modello nell'istante di tempo t_i e
• $\mathbf{t_0}$ e t_n sono il primo e l'ultimo istante di tempo considerati

Allora:
• **V** consiste di n+1 sottoinsiemi disgiunti **$V(t_0),\ldots,V(t_n)$**

...BN Dinamici: una definizione generale



E è l'insieme degli archi orientati:
E={(v,u) |
v in V(t_{i-1}), u in V(t_i), con i<=n}

cioè è l'insieme degli ***archi temporali***
relativi alle fasce di tempo t_i,
con i = 1, ... n-1

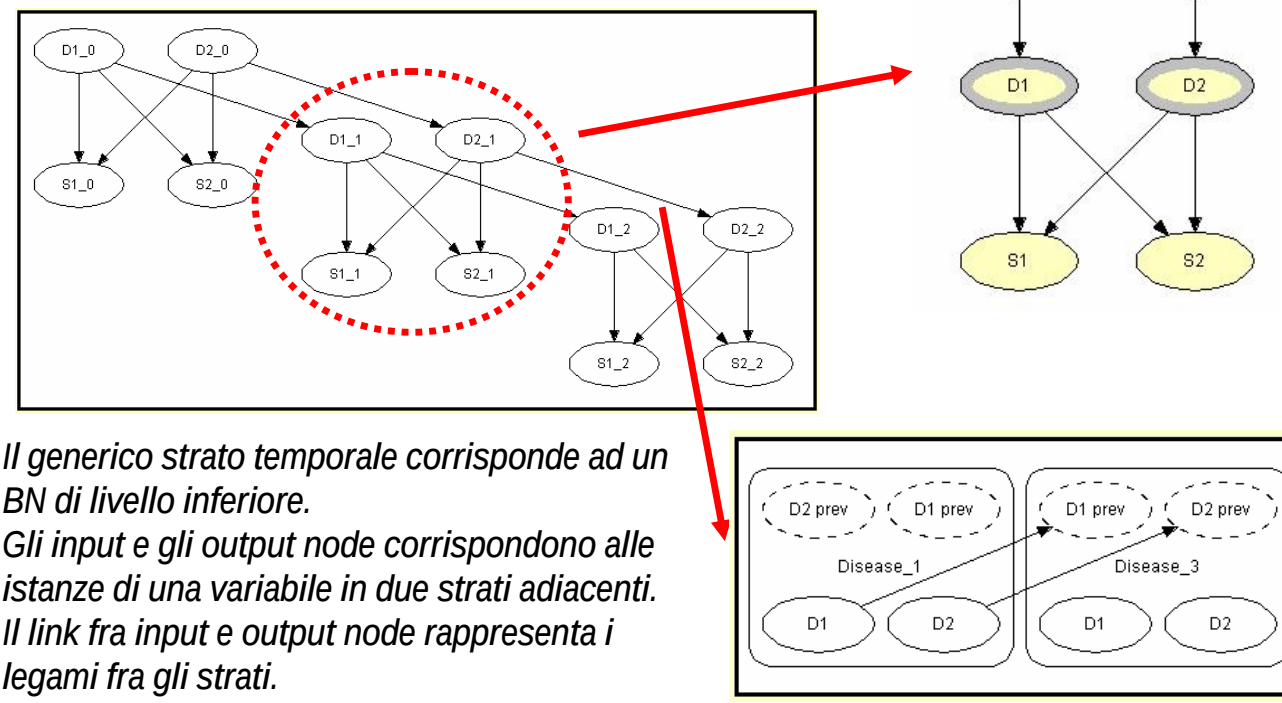
Gli archi temporali definiscono
come le distribuzioni delle variabili al tempo t_i sono condizionalmente dipendenti dalla loro distribuzione al tempo t_{i-1}.

Belief Network Dinamici (Hugin Web site)

- An *Object-Oriented Network* is a network (i.e., Bayesian network or influence diagram) that, in addition to the usual nodes, contains *instance nodes*.
- An *instance node* is a node representing an instance of another network. In other words, an instance node represents a subnet.
- Therefore, following standard object-oriented terminology, an object-oriented network is often referred to as a *class*.
- Of course, the network of which instances exist in other networks can itself contain instance nodes, whereby an object-oriented network can be viewed as a hierarchical description (or model) of a problem domain.

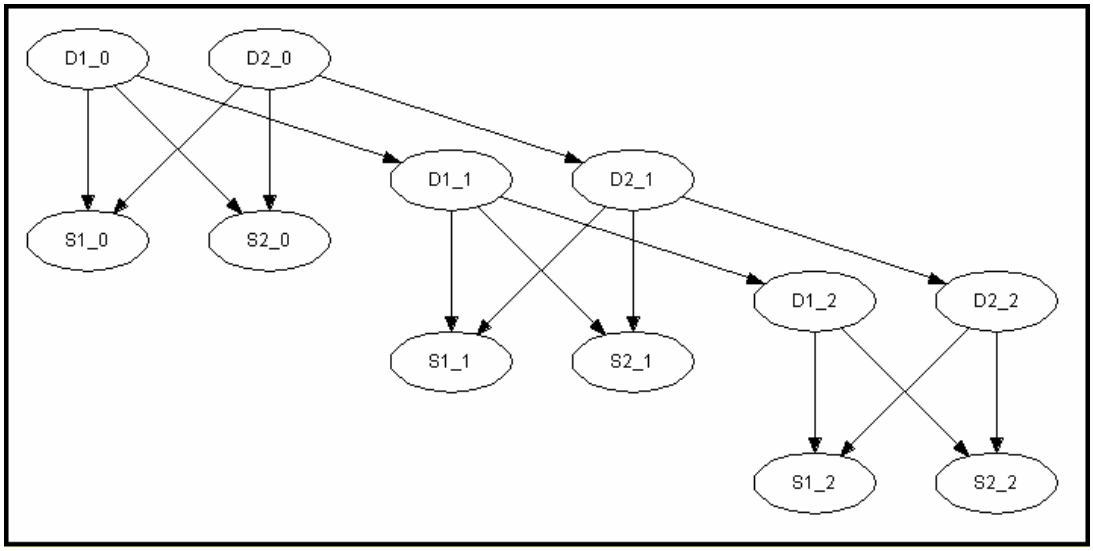
DBN come BN gerarchici in Hugin

In Hugin, il collegamento fra BN a diversi livelli di astrazione è realizzato attraverso la creazione di 'instance BN' e
il collegamento di 'input nodes' a 'output nodes'.



Il generico strato temporale corrisponde ad un BN di livello inferiore.
Gli input e gli output node corrispondono alle istanze di una variabile in due strati adiacenti.
Il link fra input e output node rappresenta i legami fra gli strati.

Costruiamo una DBN: obiettivo

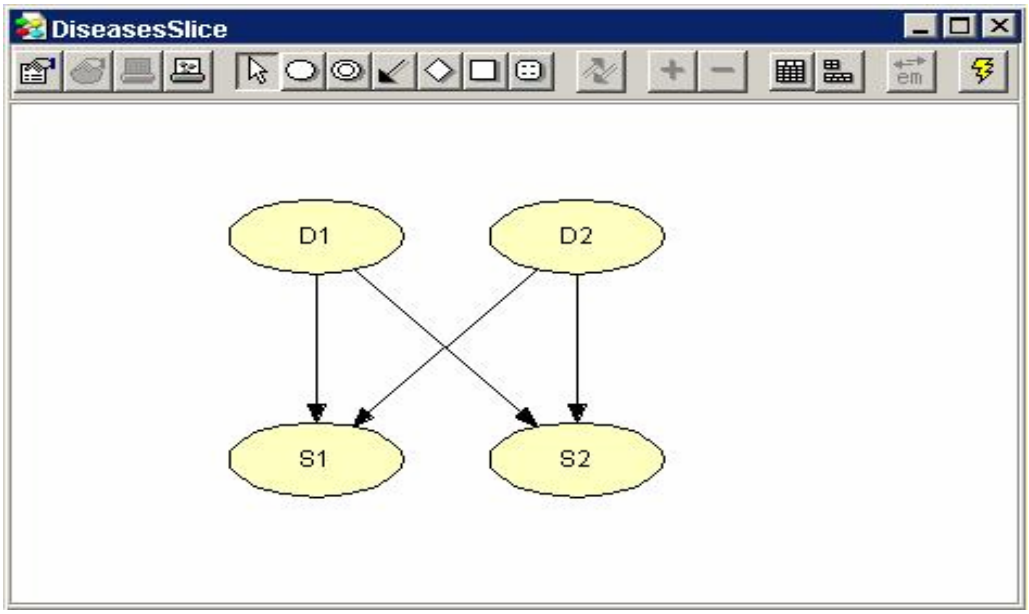


Rappresentazione mediante BN del Diseases problem.

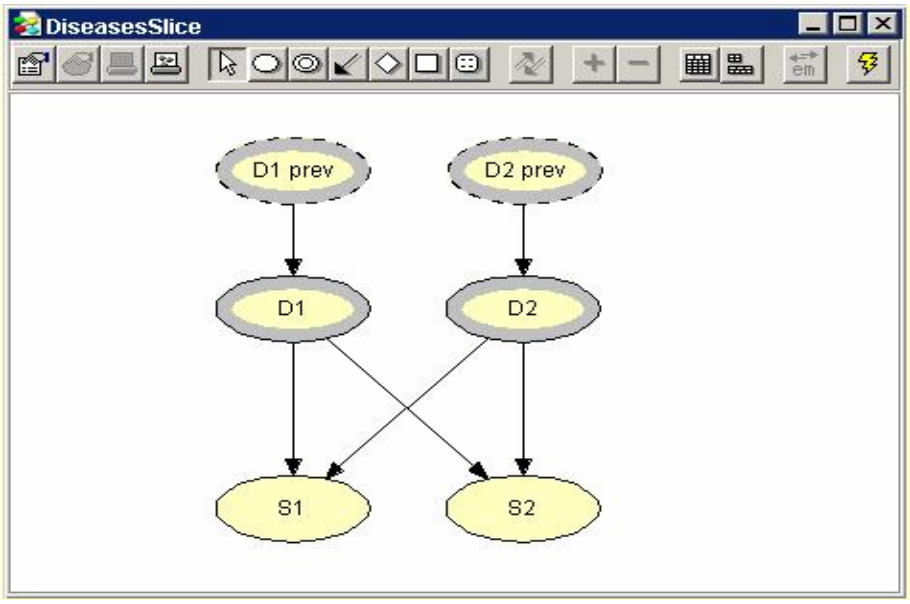
D1 e D2: due differenti malattie

S1 e S2: i sintomi che possono essere osservati come conseguenza di entrambe le malattie.

Costruiamo una DBN:
Creazione della singola sottorete (time slice)



Costruiamo una DBN:
Creazione dell'Interface Node



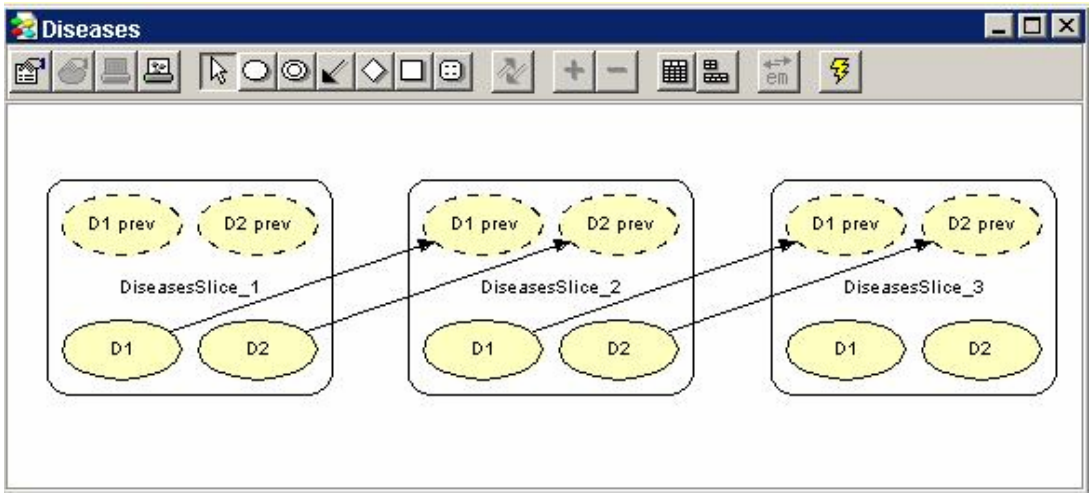
D1 e D2: Output node ottenuti spuntando l' "Output" check box nel Node Properties pane per ognuno di essi

"D1 prev" e "D2 prev" : input nodes, ottenuti creando due nuovi nodi e spuntanto l' "input" check box nel Node Properties Pane per ognuno di essi.

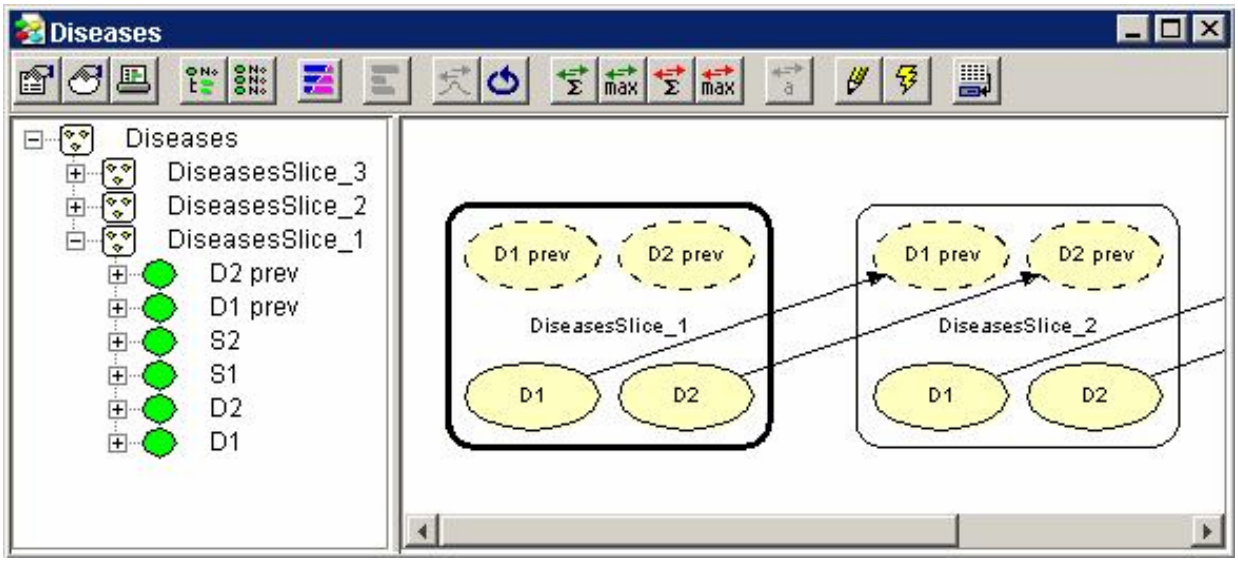
Attenzione: i nodi "D1 prev" and "D2 prev" sono *placeholder nodes* per D1 and D2, rispettivamente, nella sottorete immediatamente precedente. I placeholder nodes sono *input nodes*, e non devono essere confusi con I nodi reali!!!

Costruiamo una DBN:
Creazione del Diseases model

1) Creare una nuova rete vuota
2) Selezionare Instance Tool e creare tre instance nodes nel network pane
3) Linkare gli *output nodes* degli istance node al tempo precedente con gli *input nodes* degli instances node al tempo successivo



Costruiamo una DBN:
Running the Object-Oriented Network



Consideriamo un altro esempio nel dominio della persuasione

Esempio3

OOBN

Rappresentazione in un meta-linguaggio (molto simile a java) delle reti Bayessiane

Possibilità di trattare le reti Bayesiane come classi quindi collegare la potenza di java (metodi, attributi, ereditarietà) alla gestione e utilizzo delle reti.

Fasi di creazione:

- Progettare la rete con Hugin e salvare la rete in formato OOBN (non salvare la rete se è stata propagata).
- Preparare la classe che accederà ed elaborerà le reti SEGUENDO QUESTE FASI:
  - Definire un ClassCollection (ClassCollection cc = new ClassCollection())
  - Richiamare il metodo parserClasses di cc per leggere i file oobn (cc.parserClasses (“nomefile.oobn”)) a questo punto le reti opportunamente convertire in classi sono presenti nel cc il quale eredita i metodi della classe vector e va gestito come iun vettore.

Usare le reti:

Per usare le reti precedentemente elaborate occorre definire un dominio in cui queste reti vanno elaborate:

- Domain domain = new Domain();
- Istanziamo il dominio con domain = test.createDomain();
- domain.trinagulate (Domain.H_TM_FILL_IN_WEIGHT);
- domain.compile

OOBN 2

Usare le reti:
Per usare le reti precedentemente elaborate occorre definire un dominio in cui queste reti vanno elaborate:

- Domain domain = new Domain();
- Istanziamo il dominio con domain = test.createDomain();
- domain.trinagulate (Domain.H_TM_FILL_IN_WEIGHT);
- domain.compile

A questo punto il dominio contenente le reti può essere elaborato in quanto
Ore è possibile riferirsi ad un qualunque nodo della rete per settare una evidenza
Oppure leggere un valore o modificare un peso percentuale

Terminata la preparazione delle reti occorre propagare, semplice:

domain.propagate (Domain.H_EQUILIBRIUM_SUM,
Domain.H_EVIDENCE_MODE_NORMAL)

Dopo di che possiamo interrogare i nodi per osservare il risultato della propagazione

OOBN dinamiche

Si intende il caso in cui vogliamo collegare due o più reti oobn ottenendo una Struttura ad albero complessa partendo da reti semplici.
Rimane valido tutto quello detto fin'ora, la differenza è nell'identificazione dei nodi Di collegamento che definiremo nodi di Input e di Output.

Le fasi da seguire sono:

- inserire tutte le classi ottenute dalle reti oobn in un vettore di classi hugin definito in questa maniera : COM.hugin.HAPI.Class[ ] vetclass = new COM.hugin.HAPI.Class[10]
- l'inserimento avviene creando l'oggetto ClassList cL = cc.getMembers() e poi navigandolo prendendo ogni classe del class list e inserendola nel vettore
- dalla rete "father" occorre identificare il nodo di output questo nodo verra chiamato actualNode per convenzione e dalla rete figlio identificare il nodo di input formalNode
- ora creamo un IstanceNode usando la rete figlio ed inserendola nel class collection home: IstanceNode instnode = new IstanceNode (home, 'elemento del vettore delle classi contenente la rete figlio)
- Ultimo step settaggio dell' input: instnode.setInput(formalNode, actualNode)

Nel caso in cui la struttura è più complessa il processo va ripetuto.
Ora instanziare il dominio come visto e il gioco è fatto.

Riferimenti

- **http://www.hugin.com/**
- **http://www-2.cs.cmu.edu/~javabayes**
- **http://www.norsys.com/**
- **http://www.lumina.com/**