**Inference overview** –

This module provides a high-level overview of the main types of inference tasks typically encountered in graphic models: conditional probability queries, finding the most likely assignment (MAP inference).

**Inference overview** – Conditional Probability Query

Conditional Probability Queries

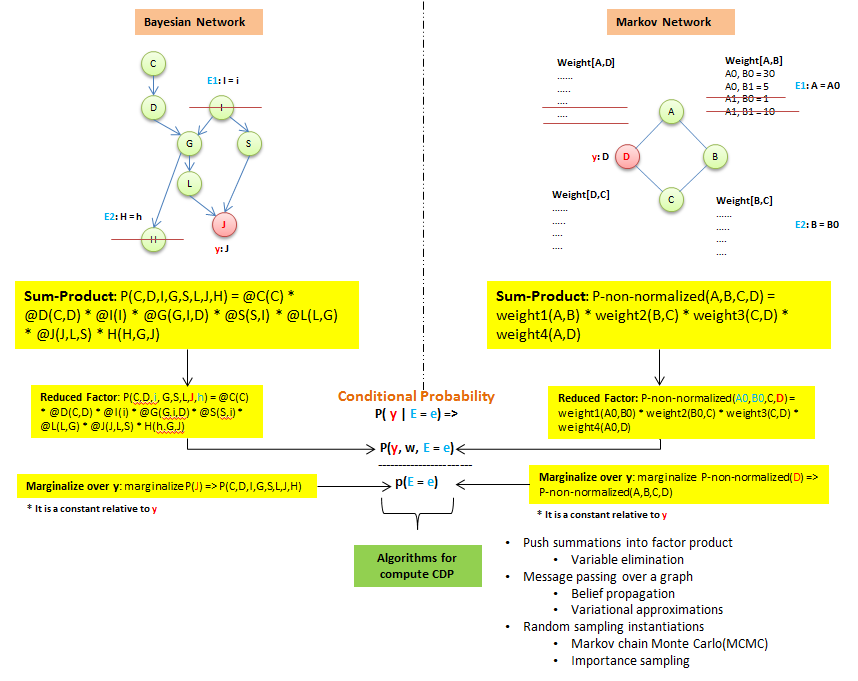
**Evidence**: **E** = e

**Query**: a subset of variables **y**

**Task**: compute **P(y | E = e)**

***Example*:** *Given value of certain test result(E=e) what is the probabilities of each disease p(y | E = e)?*

*\*\* NP Hardness – No efficient way to solve the problem*



**Inference overview** – MAP inference

MAP – Maximum a Posteriori

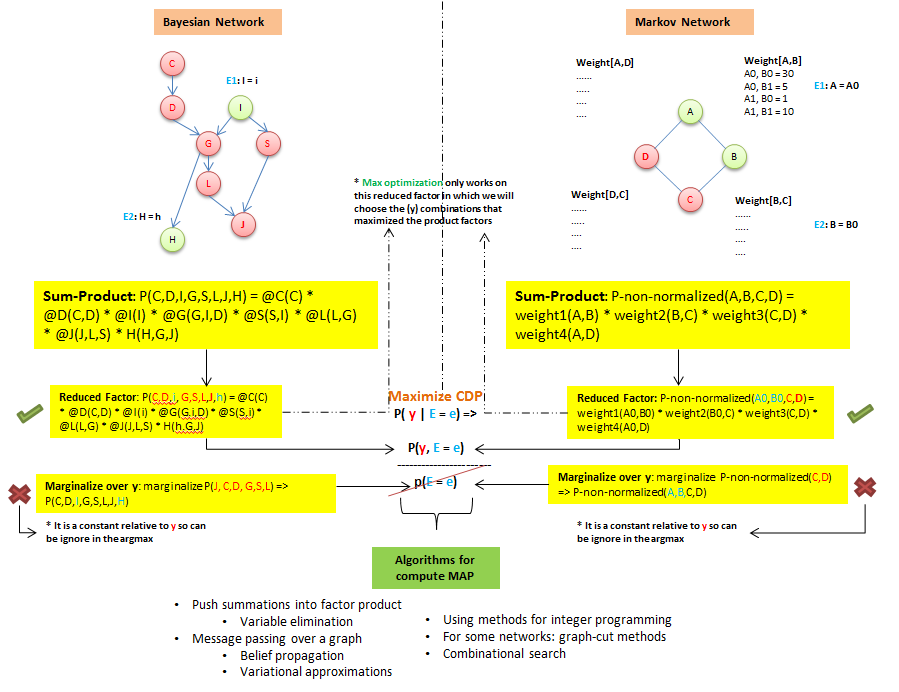
**Evidence**: **E** = e

**Query**: all variables **y** which are not in **E**

**Task**: **MAP(y | E = e)** => **argmax(y) P(Y=y | E=e) \***the combination**(y)** that gives max posteriori probability (Could be multiple ones)

***Example*:** *Image segmentation, the most likely segmentation max-> p(y | E = e)?*

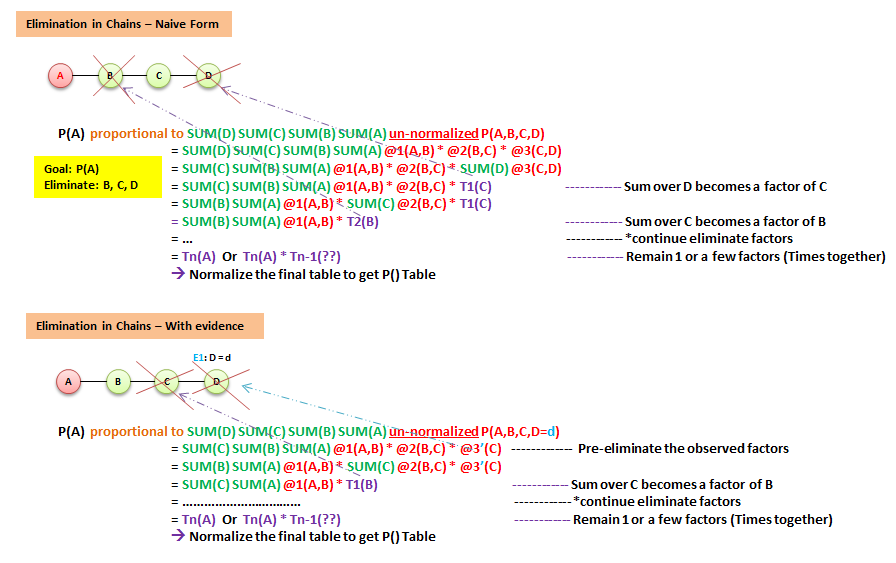
*\*\* NP Hardness – No efficient way to solve the problem*



**Algorithms for compute inference**

**Algorithms for compute inference** - variable elimination

This module presents the simplest algorithm for exact inference in graphic models: variable elimination.

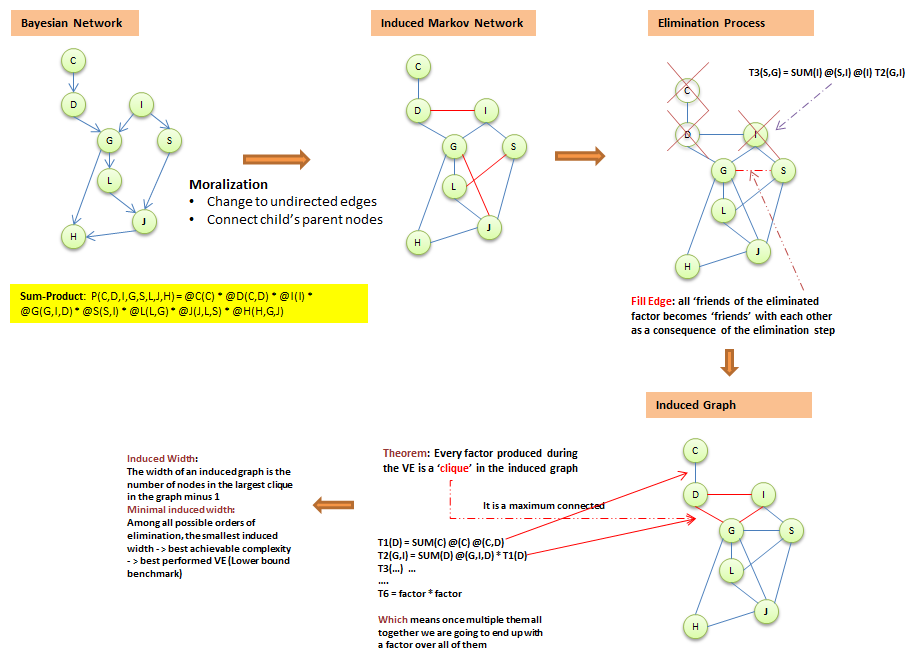


Algorithm complexity:

* It is linear in size of model (# factors, # variables), size of the largest factor generated
* Size of factor is exponential in its scope – drives complexity

Which means the choice of elimination is important!

Graph-Based Perspective on Variable elimination –



**How to find the optimal ordering of variable elimination which generates simplest Induced Graph?**

Method1:

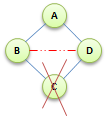
At each point, eliminate node with smallest cost ->

Possible cost functions:

* **Min-neighbors**: # neighbors in current graph
* **Min-weight**: # of values in the factors (2 classes vs 200 classes)
* **Min-fill**: number of new filled edges after elimination
* **Weighted min-fill**: # of values of the new filled edges

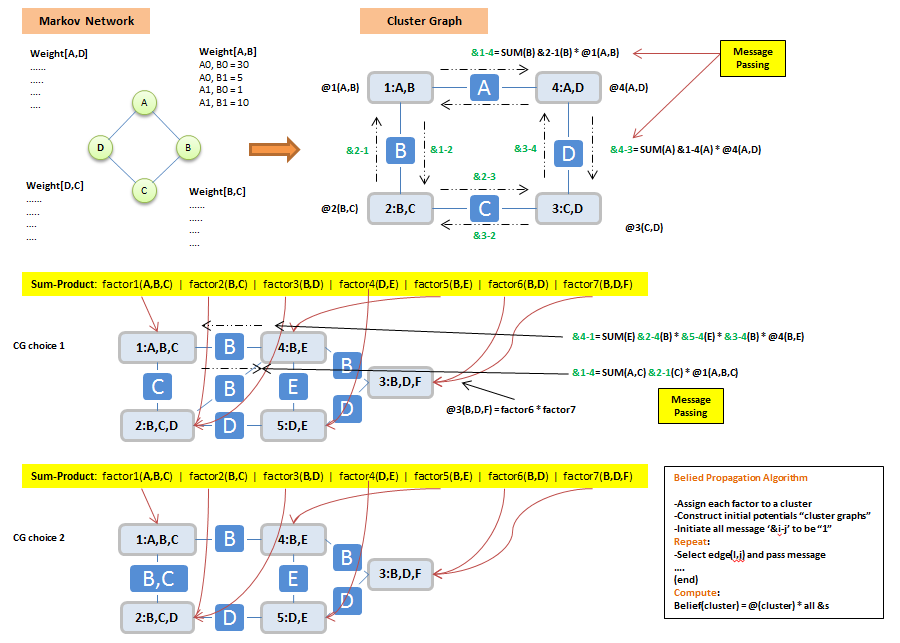
Method2:

**Theorem** – The induced graph is triangulated (No loops of length > 3 without a “bridage”



Find an elimination order by finding a low-width triangulation of original graph

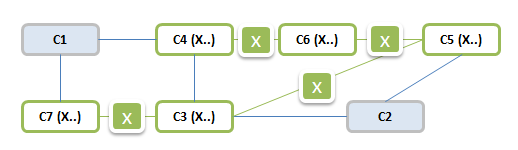
**Algorithms for compute inference** – Belief Propagation Algorithms (Message passing)



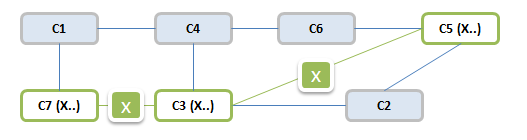
**Cluster Graph Properties -**

**Family Preservation** – Given a set of factors, we assign factors to a set of clusters -> For each factor, there exists a cluster that its subset equals to the factor.

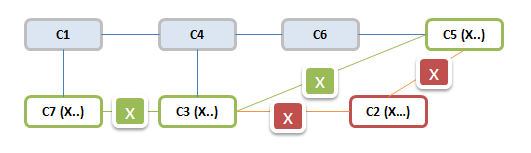
**Running Intersection Property** - For any X, the set of clusters and sepsets containing form a **tree**



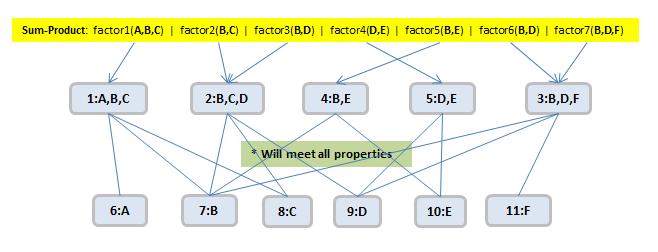
1: For any pair of cluster i and j, if they have factor x in common, there is definitely a unique path contains x between cluster 1 and cluster j all contains X.



2: No feedback loop is allowed. It will extremely skewed the probability



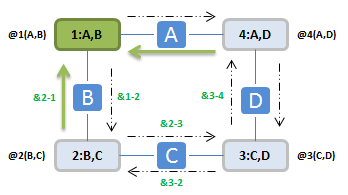
How to create a cluster graph – choice 1 – **Beta Cluster Graph**



**Belief Propagation Properties –**

**Cluster Beliefs:**

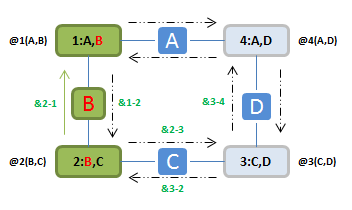
**Belief**(**Cluster1(A,B)**) = @1(A,B) \* &4-1(A) \* &2-1(B)



**Calibration:**

A cluster graph is **calibrated** if every pair of adjacent cluster I and j agree on their sepset (X)

Marginalize(B) **Belief**(**cluster1(A,B)**) = Marginalize(B) **Belief**(**Cluster2(B,C))**

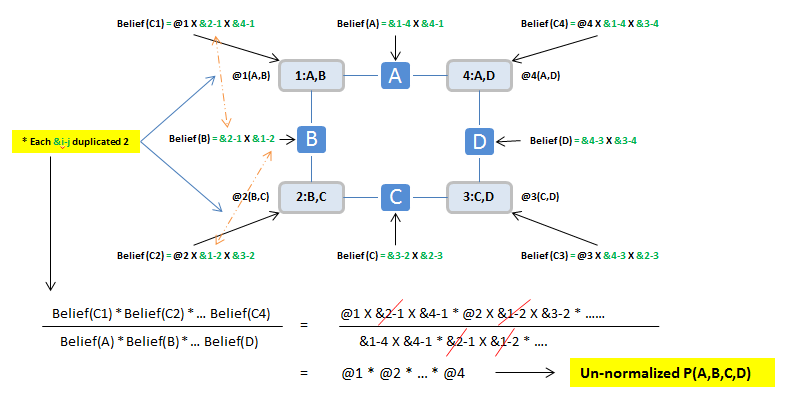


**Convergence** implies **Calibration ->**

**Convergence:**

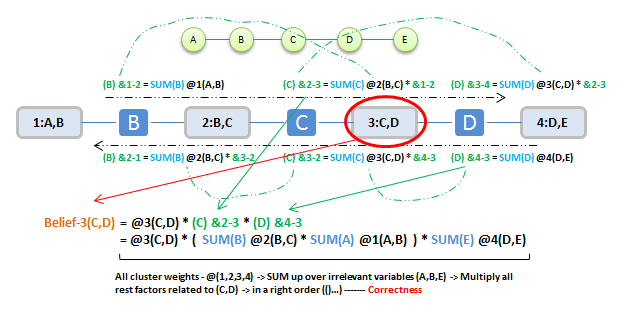
**&i-j(X)[**current iter**] = &i-j(X)[**previous iter**]**

**Re-parameterization**:



**Clique Trees – Algorithm (Correctness)**

**Correctness:**

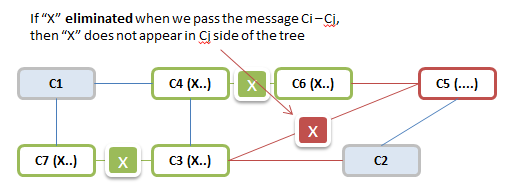
****

**Clique Tree:**

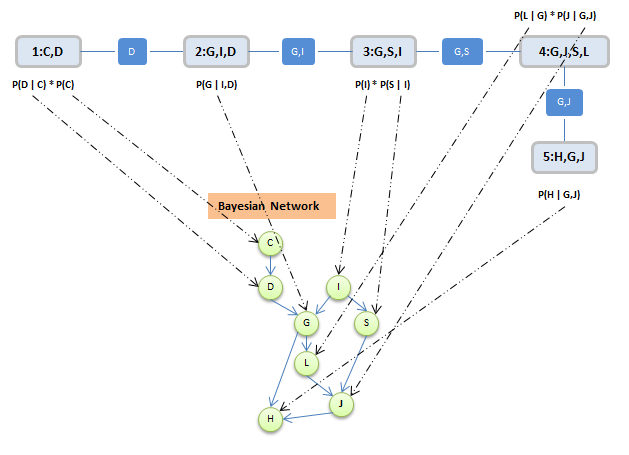
**Exactly like cluster graph but use all joint subset as edge between clusters.**

**Properties –**

* **Family Preservation**
* **Running Intersection Property**

****

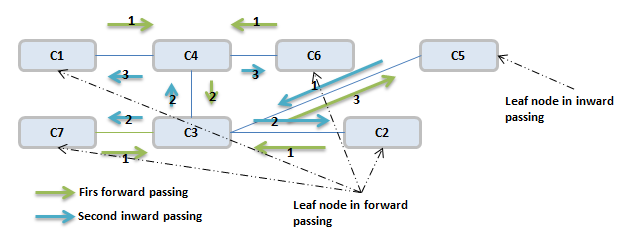
**More complex example of Clique Tree:**

****

**Clique Trees – Algorithm (Computation)**

**Passing message orders:**

* All leaf node immediately finalized and passing message
* From leaf node forward and inward totally 2(k-1) message passing performed, K = 3 edges, which is enough to compute all beliefs.
* Once all Ci neighbors finalized sending message, then & i-j is finalized
* Can compute marginal over all variables at only twice the cost of variable elimination
* It stores the messages, inference can be reused in incremental queries



**Answering Queries:**

**What is the posterior distribution on variables that appear together in clique?**

* **Sum out irrelevant variables from any clique containing those variables**

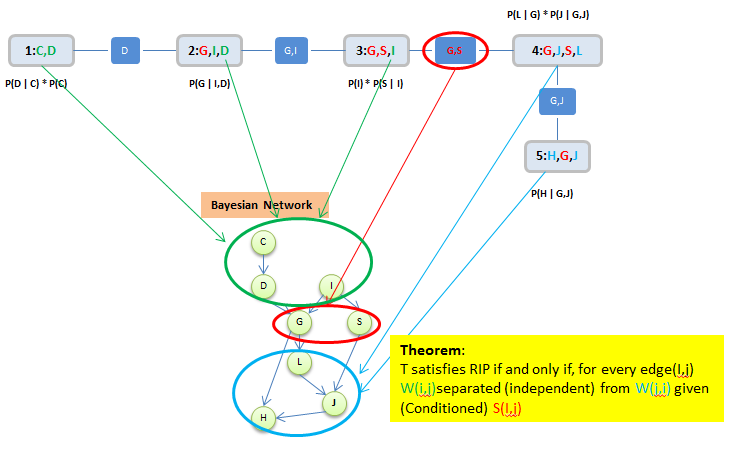
**If introducing new evidence Z=z and querying X variable?**

* **If X appears in clique with Z, then we can condition / reduce by I(Z=z) like @(Z=z,X), then sum out irrelevant variables and renormalize**
* **If X NOT appears in clique with Z, then we can condition / reduce by I(Z=z) in cliques that contains Z and then multiply over other cliques along the way to cliques contains X, then other cliques keep the same**

**Clique Trees – Algorithm (Clique Tree and Independence)**

**For an edge (I,j) in clique tree T, let:**

* **W(I,j) = all variables that appear only on Ci side of T**
* **W(j,i) = all variables that appear only on Cj side of T**
* **Variables on both sides are in the subset S (i,j)**



Correctness of clique tree inference relies on RIP -> RIP implies separation in original distribution

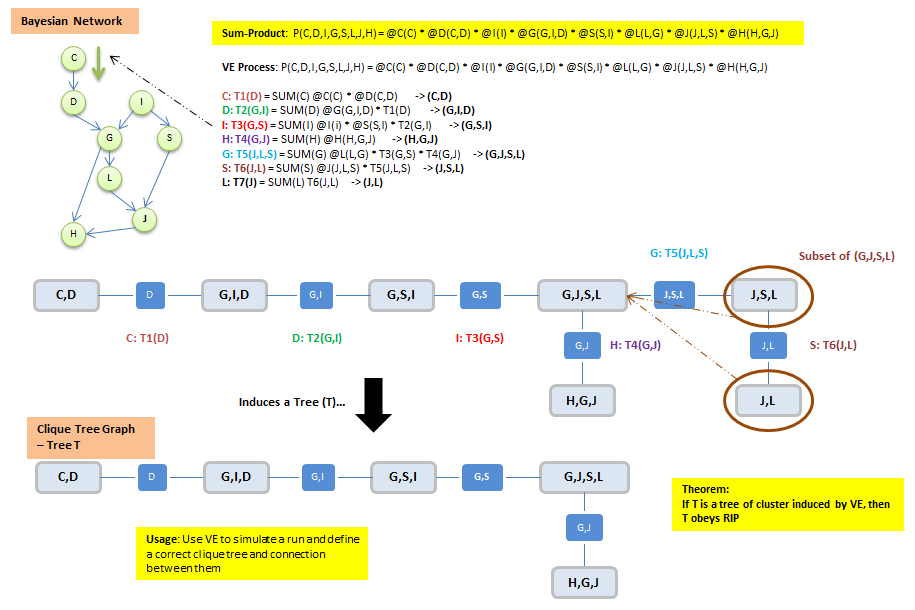
Implies minimal complexity incurred by any clique tree: related to minimal induced width of graph

**Clique Trees – Clique Tree and VE**

**VE process induces a tree:**

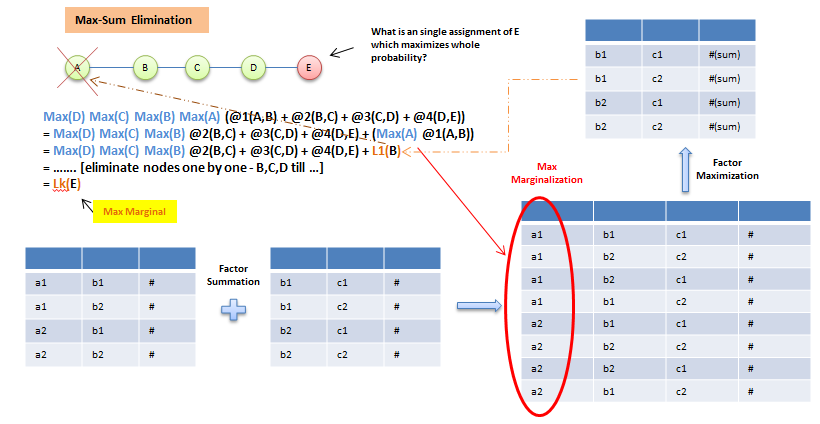
In VE, each intermediate factor is used only once. Hence, each cluster “passes” a factor (message) to exactly one other cluster.

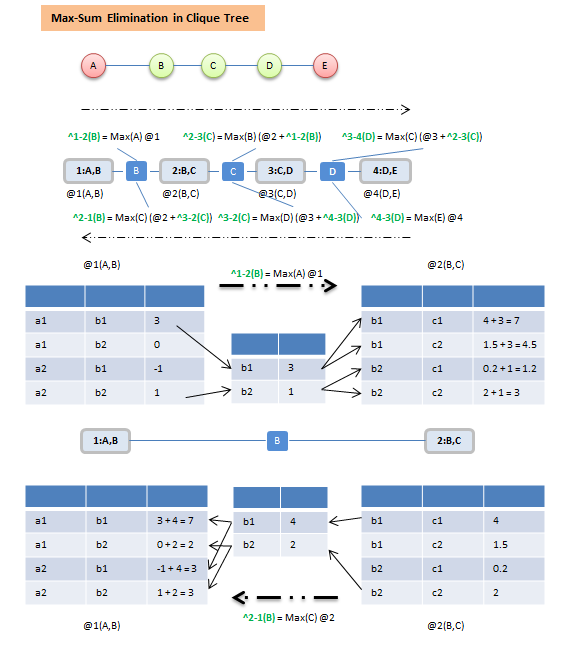
**Tree** – Family preserving, RIP



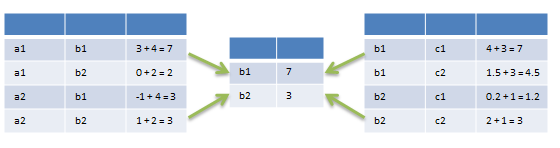
**Algorithms for compute inference** – MAP Algorithms

**Answering**: Finding a single coherent assignment that has the highest probability?





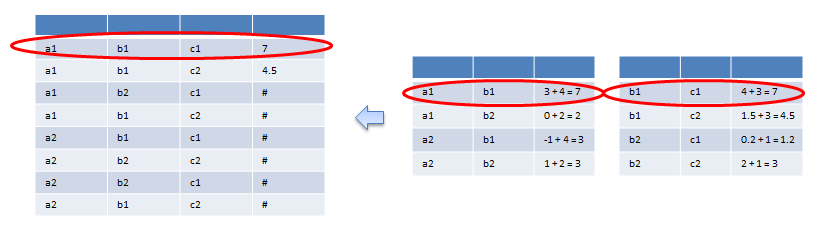
* Belief(Ci) = @i(Ci) + ^j-I … 🡪 Max Marginal of Ci (Max over all variables not in Ci)
* Calibration Property – Cliques agree on shared variables



**Algorithms for compute inference** – MAP Assignment

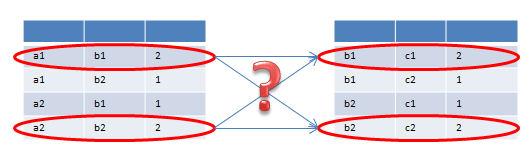
**Easy if MAP assignment is unique**:

* Single maximizing assignment at each clique
* Whose value is the @=? value of the MAP assignment
* Doe to calibration, choices at all cliques must agree



**Easy if MAP assignment is NOT unique**:

* If MAP assignment is not unique, we may have multiple choices at some cliques
* Arbitrary tie-breaking may not produce a MAP assignment



**Solution:**

* Slightly perturb parameters to make MAP unique (Add few noisy to make it unique)
* Use trace-back procedure that incrementally builds a MAP assignment, one variable at a time (Id previous B = b1, then next clique, B can only = b1, etc)

**Algorithms for compute inference** – Other MAP Algorithms

**To Be Continuous …**

**Sampling Methods** – Simple Sampling

Rather than manipulate the exponentially large distribution as whole, we randomly sample instance from that distribution and use those instance as a sparse representation of the distribution so that we can estimate the statistic quantity of the distributions.

**Sampling on continuous quantities:**

* **Weighted AVG(all sampled groups’ expectation metrics)**

**Sample on discrete quantities:**

* **X(value = a,b,c,….) => select elements by (random number generator)**

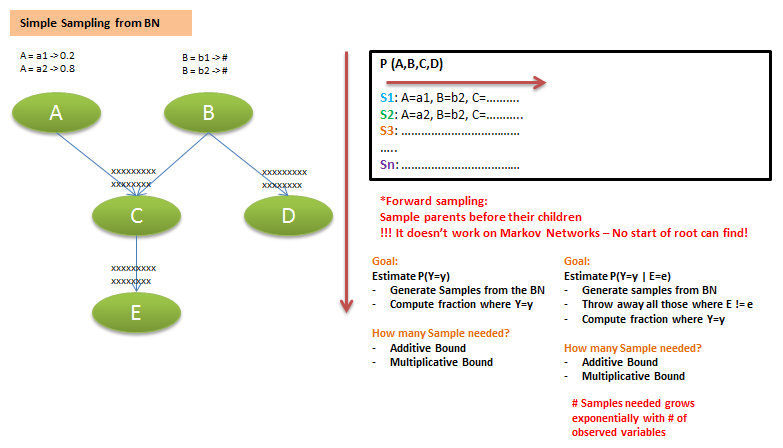
**Estimated Bound –** probability to get a bad sample**: it exists but usefulness is limited**

* **Hoeffding Bound:** For additive bound ‘e’ on error with probability > 1-&:
* **Sample size >= In(2/&) / 2\*e^2**

**! Additive bound: useless for low probability events**

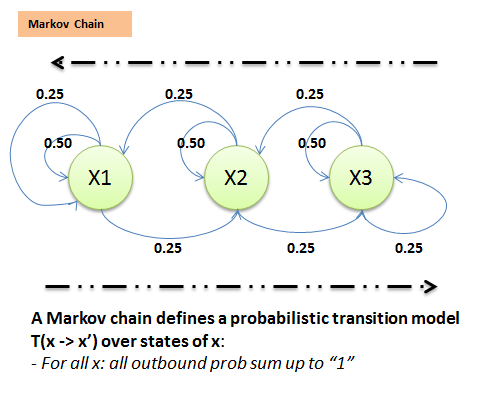
* **Chernoff Bound:** For multiplicative bound ‘e’ on error with probability > 1-&:
* **Sample size >= 3 \* (Ln(2/&) / p \* e^2)**

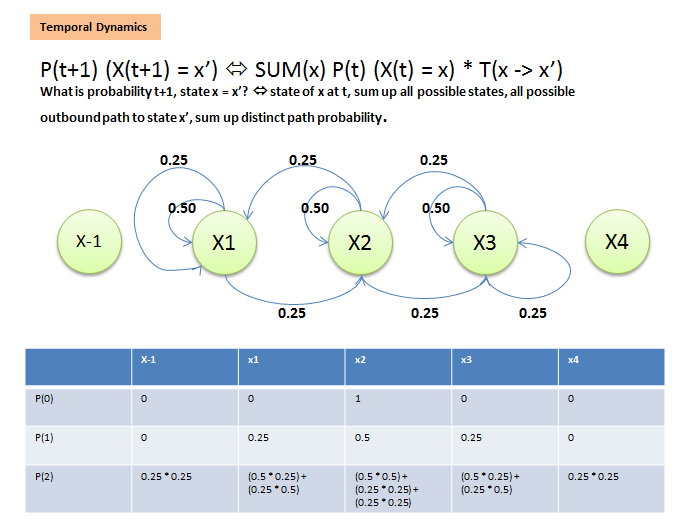
**! Multiplicative bound: # samples grows as 1/P(y: event we try to estimate)**

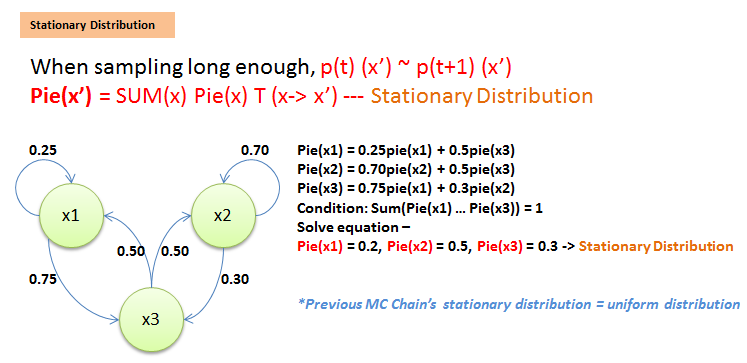
****

**Sampling Methods** – MCMC

Allows us to sample **indirectly** from distribution P.







**Regular Markov Chains** –

A **Markov chain** is regular if there is a K such that, for every x, x’, the probability of getting from x to x’ in exactly K steps is > 0

**Theorem**: A regular Markov chain converges to a unique stationary distribution regardless of start state x

**Sufficient condition for regularity**:

* Every two states are connected
* For every state, there is a self-transition

**Sampling Methods** – Using MCMC

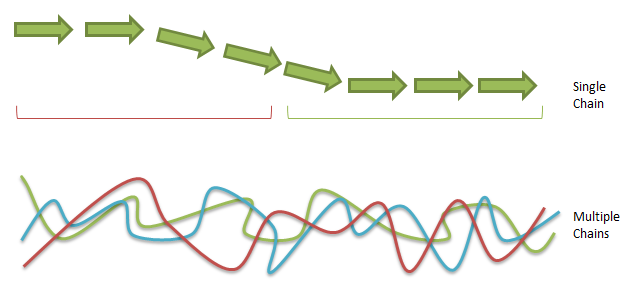
**Using a Markov Chain:**

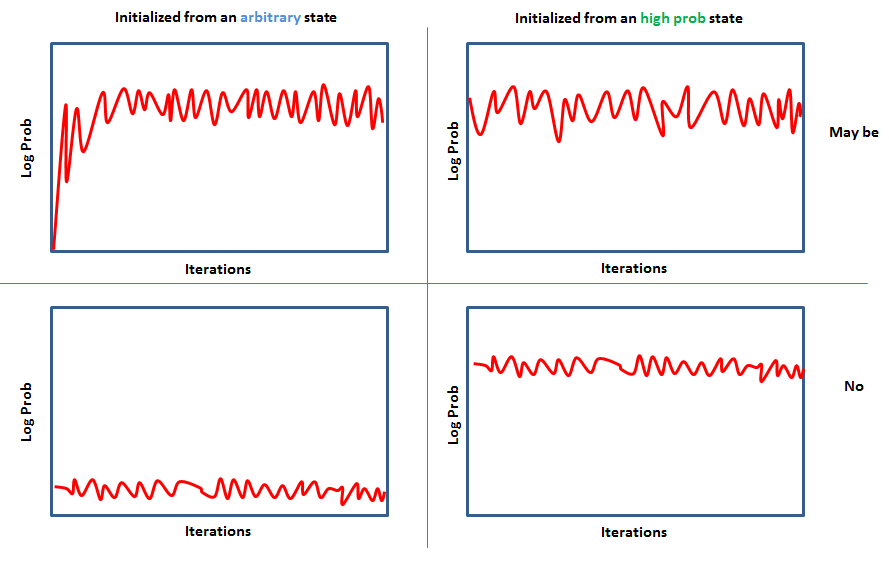
* **Goal**: compute some probability relative to distribution **P** (But P is too hard to sample from directly)
* Construct a Markov Chain T whose unique stationary distribution is **P**
* **Sample** x(0) from P(0)
* For T = 0, 1, 2, …
  + Generate x(t+1) from T(x(t) -> x’)

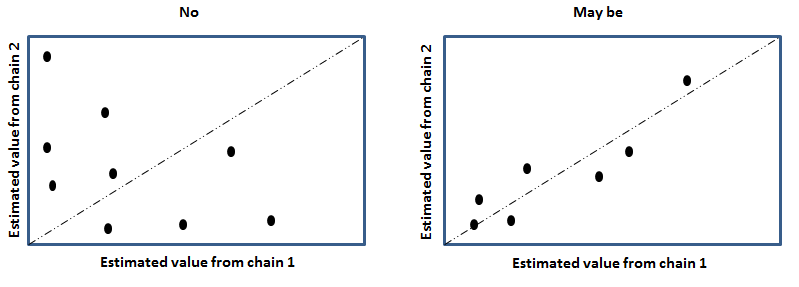
Until When can use the sample? - > Run long enough to “**Mix**”:

How to proof a chain is “**Mixed**”? (You can’t but you can somehow prove it has **NOT**)

* Compare chain **statistics** in different windows within a single run of chain and across different runs initialized differently







**Once the chain mixed**:

* Use all samples x(i) are from the stationary distribution, however, near sample tends to correlated to each other
* The longer to wait till **mix**, the more correlated the samples

**MCMC Algorithm Summary I**

**Running** C chains in parallel, For c = 1, …, C

* Sample x(c,0) from P(0)

**Repeat** until Mixing (single chain), For c=1, .. ,C

* Generate x(c, t+1) from T(x(c,t) -> x’) – random walk

**Compare** window statistics in different chains to determine mixing

Till mixing, repeat sample until sufficient samples

* Start with empty sample set
* For each chain, c = 1,…,C
  + Generate x(c,t+1) from T(x(c,t) -> x’)
  + Add up sample set

Once, got sample distribution set, get all attributes

**Pros**: Very general purpose, often easy to implement (local sampling), Good theoretical guarantee as sample large enough

**Cons**: Lots of tunable parameters/design choice, slow to converge, difficult to tell whether a chain is working(Mix?)

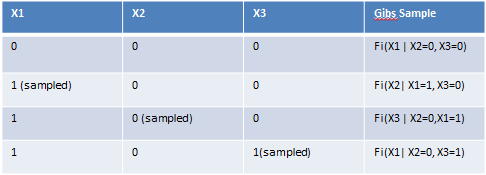
**Sampling Methods** – Gibs sampler

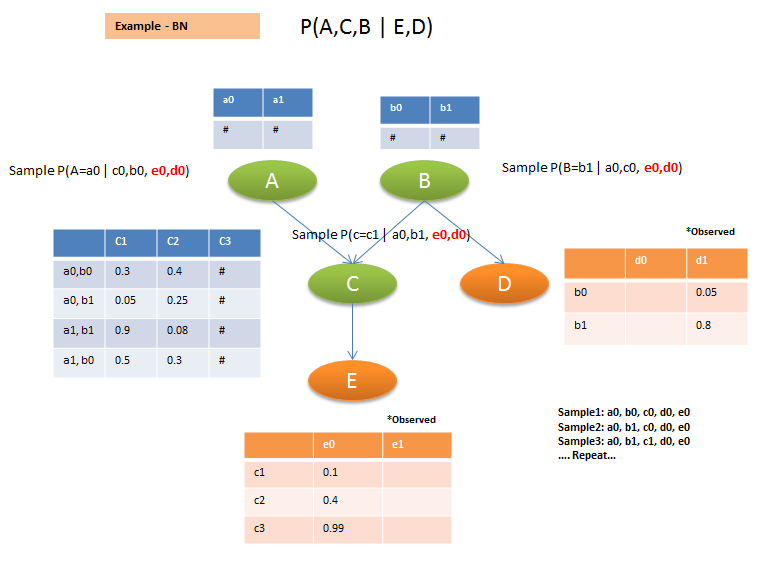
Target distribution – **Gibs distribution**: fi(X1,…,Xn)

Markov chain state space: complete assignments x to X = {X1, …, Xn}

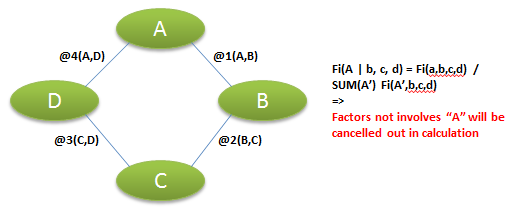
Transition model given starting state **x**:

* For i=1, …, n
  + Sample x(i) ~ fi(x(i) | x(-i)) – Condition on current non-ith Xs, sample x(i)
  + Set x’ = x, move to next i+1





**Computational Cost: Example in MN**

****

**Gibbs Chain and Regularity** – Sample long enough will converge

* If all factors are positive, Gibbs chain is regular
* But regular doesn’t mean good, can still be very slow to converge

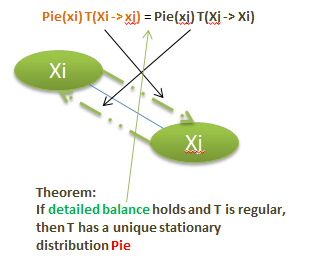
**Pros**: The simplest Markov Chain for PGMs, computationally efficient to sample

**Cons**: Often slow to mix, when a lot of peak probabilities, Only applicable to sample small discrete factors (Product factors) – Not densely connected or continuous values

**Sampling Methods** – Metropolis – Hastings Algorithm

Improve Gibbs sampler by allowing for designing a Markov chain with desire stationary distribution. For a given stationary distribution, we can design a whole family of different Markov chains that explore the space differently and then select one amount those has good convergence.

**Reversible Chains:**



**Metropolis Hasting Chain:**

Proposal distribution **Q(x -> x’)** ---- Freely over the space

Acceptance probability **A(x -> x’)** ------ How good is this move?

**At each state x, sample x’ from Q(x -> x’)**

**Accept proposal with probability A(x -> x’)**

* **If proposal accepted, move to x’**
* **Otherwise stay at x**

**T(x -> x’) = Q(x -> x’) A(x -> x’) if x’ <> x**

**T(x -> x) = Q(x -> x) + SUM(X=x’) Q(x -> x’) (1-A(x -> x’)**

**Acceptance Probability:**

**Detail balance:**

**Pie(x) T(x -> x’) = Pie(x’) T(X’ -> X)**

**Pie(x) Q(x -> x’) A(x -> x’) = Pie(x’) Q(x’ -> x) A(x’ -> x)**

**=>**

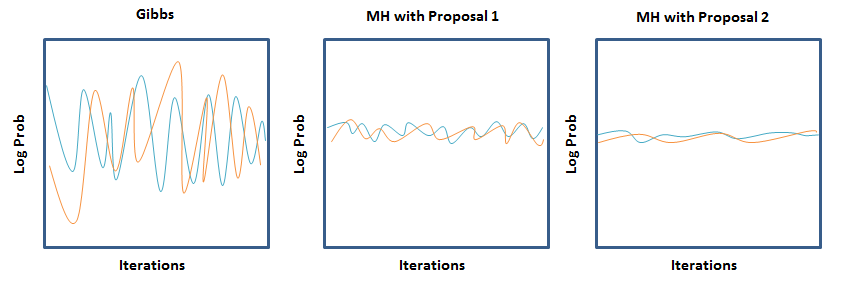
**A(x -> x’) / A(x’ -> x) = Pie(x’) Q(x’ -> x) / Pie(x) Q(x -> x’) = P < 1**

* **Make A(x -> x’) = P, A(x’ -> x) = 1**
* **A(x -> x’) = min [1 \* Pie(x’) Q(x’ -> x) / Pie(x) Q(x -> x’) ]**

**Choice of Q():**

**Not easy –**

* **Q Must be reversible:**
  + **Q(x -> x’) > 0 => Q(x’ -> x) > 0**
* **Opposing forces**
  + **Q should try to spread out, to improve mixing**
  + **But then acceptance probability often low**

****

**Summary:**

**MH is a general framework for building Markov chains with a particular stationary distribution**

* **Requires a proposal distribution**
* **Acceptance computed via detailed balance**

**Tremendous flexibility in designing proposal distributions that explore the space quickly**

* **But proposal distribution makes a big difference**
* **And finding a good one is not always easy**

**Sampling Methods** – Sampling methods summary

Which inference method should we go with?

* **Computing Marginal** –
  + **Less Fragile (**Not single a hypothesis, compute probability distribution among a range of options, is our top option a little more likely than our second option? Determine how robust our inference are**)**
  + **Confidence in answer (**See above**)**
  + **Supporting decision making (**When integrated with utility model**)**
    - **If run in approximate inference –** 
      * Errors are often attenuated (Robust answer – error on one piece not significant impact)
  + **Marginal Algorithms –** 
    - Exact inference (Good if fits memory)
    - Loopy message passing (Approximate inference)
    - Sampling methods (Approximate inference)
* **Computing MAP –** 
  + **Coherent joint assignment (**Get the best answers to individual piece of inference problem – trade off on robustness**)**
  + **More tractable model class (**A range of model classes are more tractable – more efficient method**)**
  + **Some theoretical guarantees (**In terms of how close our estimate to the correct answer**)**
    - **If run in approximate inference –** 
      * Ability to gauge whether algorithm is working
  + **Marginal Algorithms –** 
    - Exact inference (Good if it fits memory – low tree width, regular potentials)
    - Optimization methods (based on optimization – approximate methods, estimate performance)
    - Search-based methods (Like hill-climbing, includes sampling, easy to implement)

**Factors to consider in approximate Inference**:

1. **Connectivity Structure** – message passing algorithm doesn’t like densely connected network, sampling are less affect by it
2. **Strength of inference** – Strong preference for certain combinations of value – impacts both message passing and sampling algorithms (difficult to move away from current position in Gibbs)
3. **Opposing inference –** It also bad when the strength is going different directions (Like, one path likely to agree while the other likely to disagree)
4. **Multiple peaks in likelihood –** Shaper peaks even more hard, not convex, hard to search optimal

**How do we address the problems above?**

1. Identify the “problem” regions in network
2. Try to make inference in these regions more exact – May cause computation, but increase performance
   1. We can put this region into one cluster if we use cluster graph
   2. We can have proposal move over multiple variables in that regions if we use sampling
   3. We can put the entire region into “slave” in dual decomposition if we are using MAP algorithms