Bayesian network internals

Inference algorithms, time series & distributed learning with Big Data

Dr John Sandiford, CTO Bayes Server

Contents

Introduction

Profile

linkedin.com/in/johnsandiford

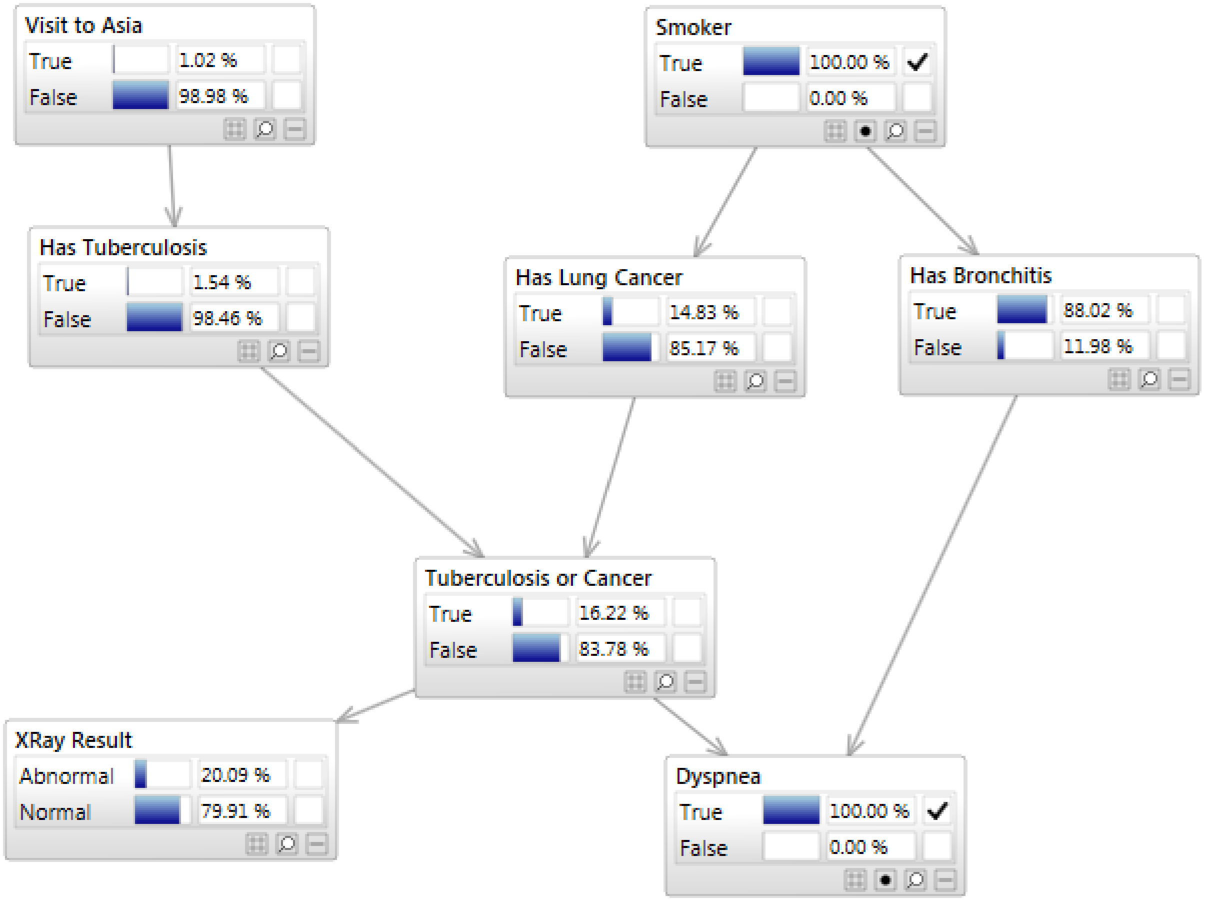
- PhD Imperial College – Bayesian networks
- Machine learning – 15 years
  - Implementation
  - Application
  - Numerous techniques
- Algorithm programming even longer
  - Scala , C#, Java, C++
- Graduate scheme – mathematician (BAE Systems)
- Artificial Intelligence / ML research program 8 years (GE/USAF)
- BP trading & risk analytics – big data + machine learning
- Also: NYSE stock exchange, hedge fund, actuarial consultancy, international newspaper

What is a Bayesian network?

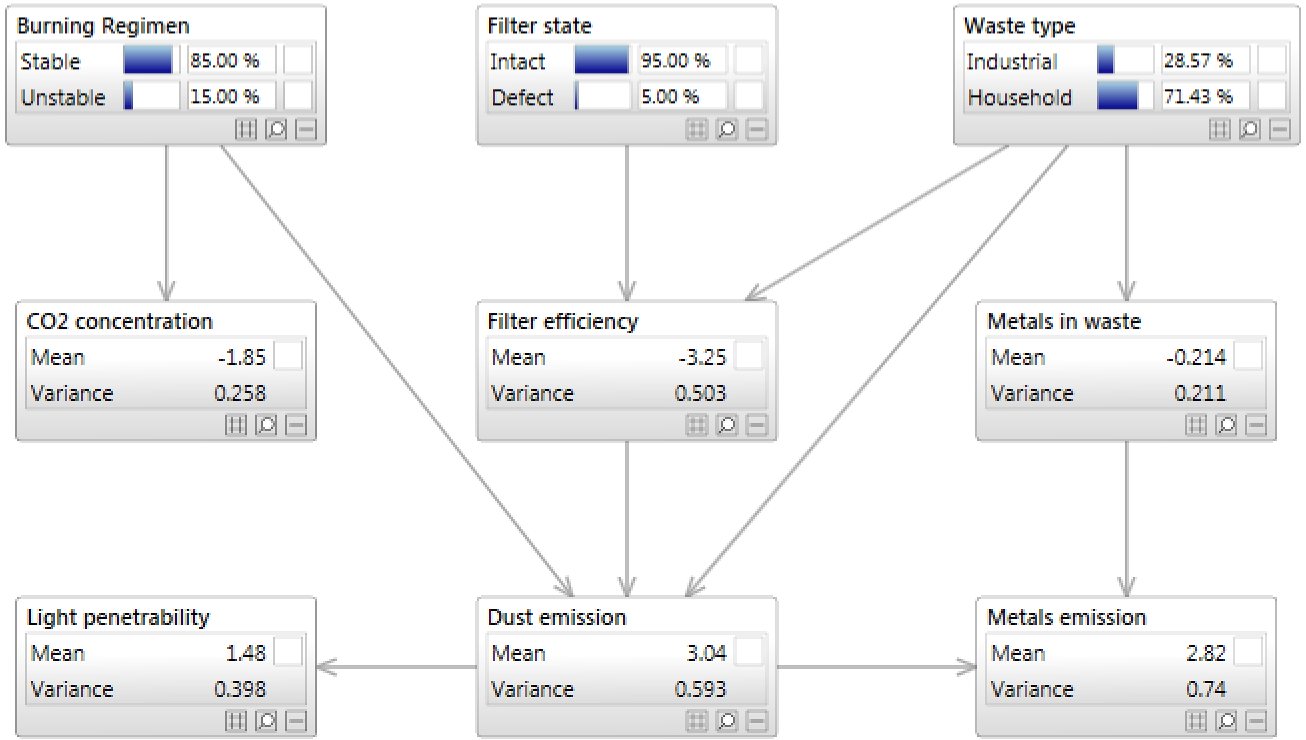
What is a Bayesian network?

- DAG – directed acyclic graph
- Nodes, links, probability distributions
- Each node requires a probability distribution conditioned on its parents (if any)

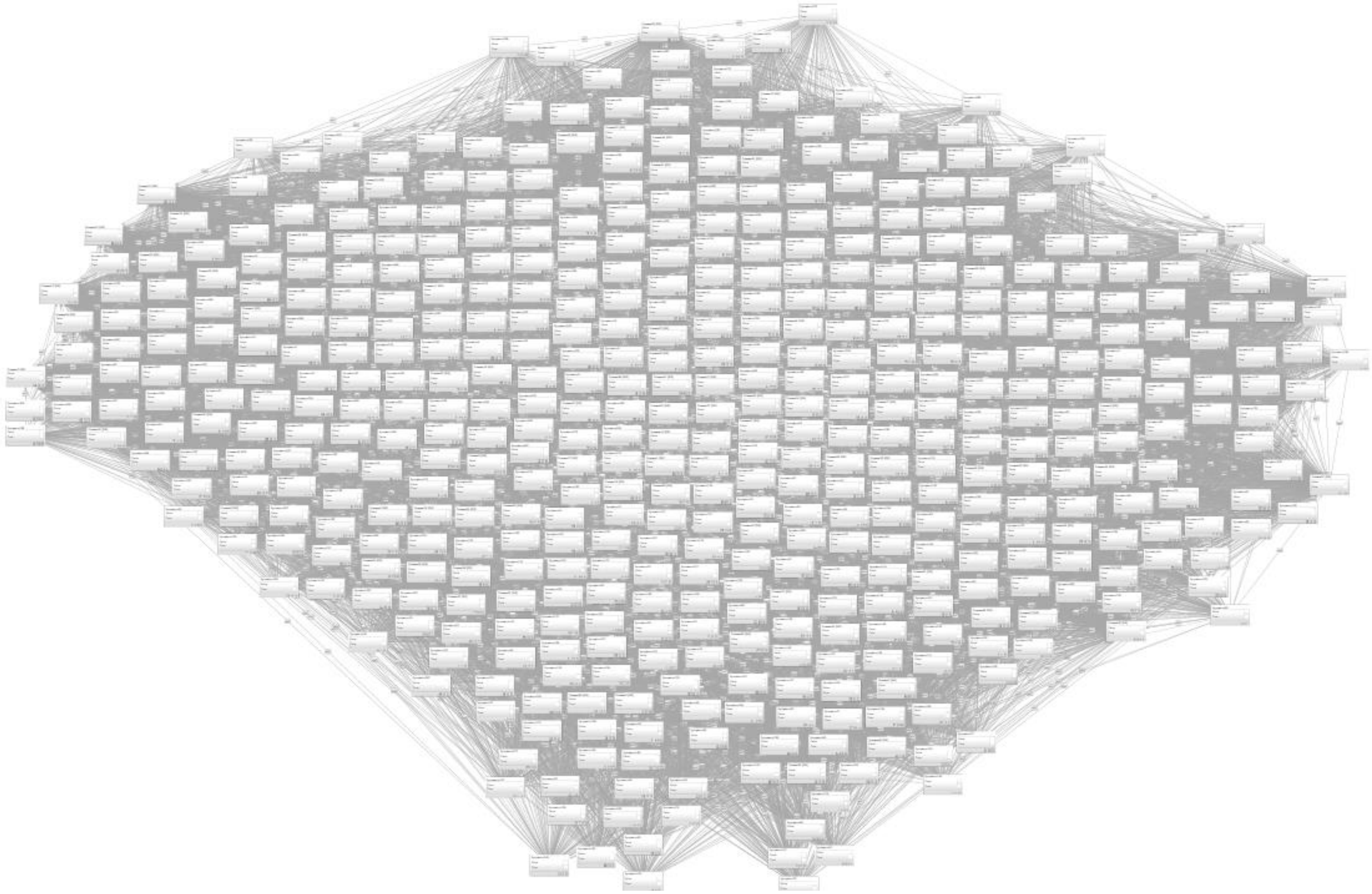
Example – Asia network



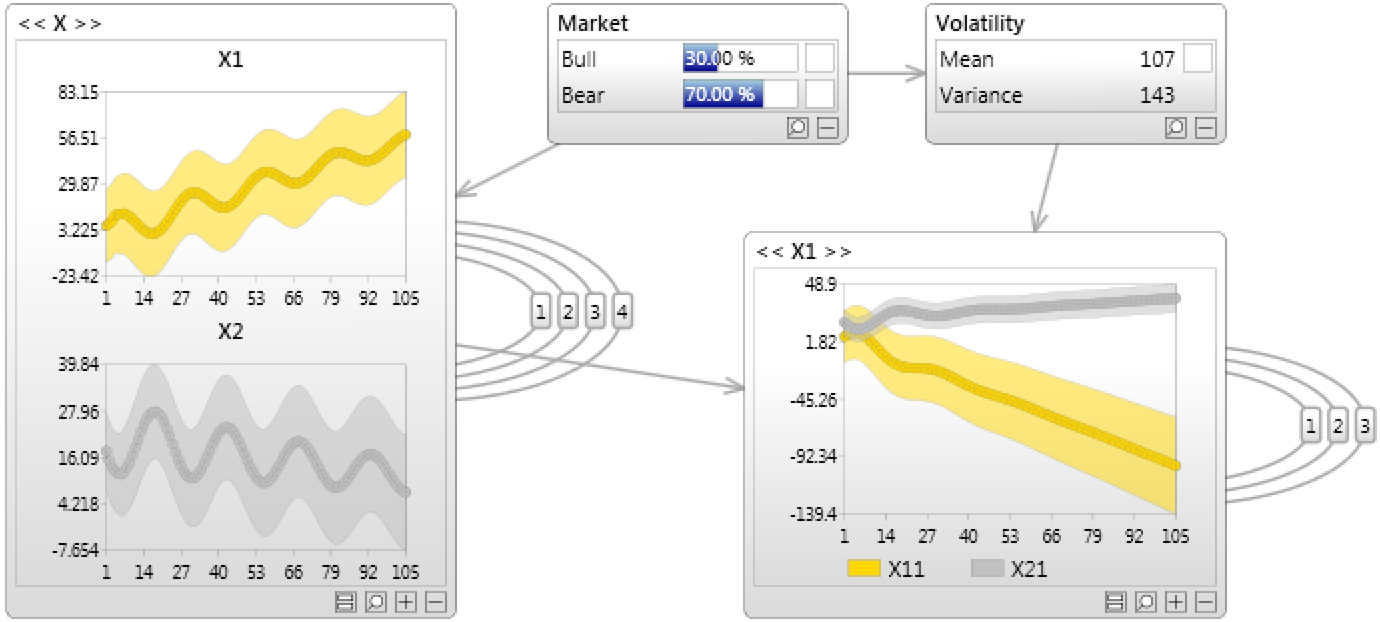
Example – Waste network



Example – the bat (40,000 links)



Example – static & temporal



Insight, prediction & diagnostics

- **Insight**
  - Automated
  - Large patterns
  - Anomalous patterns
  - Multivariate
- **Prediction / inference**
  - Supervised or unsupervised
  - Anomaly detection
  - Time series
- **Diagnostics / reasoning**
  - Troubleshooting
  - Value of information
  - Decision support

What is inference?

Inference

- Asking a question given things you already know
- Encompasses prediction, reasoning & diagnostics

- Given a number of symptoms, which diseases are most likely?
- How likely is it that a component will fail, given the current state of the system?
- Given recent behaviour of 2 stock, how will they behave together for the next 5 time steps?
- Handles missing data

Exact & approximate

- Exact inference
  - Applicable to a large range of problems, but not all
    - May not be possible when combinations/paths get large
  - Correct answer subject to rounding errors

- Approximate inference
  - Wider class of problems
  - Deterministic / non deterministic
  - No guarantee of correct answer

Exact inference

- We will discuss exact inference
- Many concepts apply to both

Probability

Probability notation

- P(A)
- P(A|B) – Conditional probability *(probability of A given B)*
- P(A,B) – Joint probability *(probability of A and B)*
- P(**Head** | **Tail**)
  - variables on the left are referred to as head, and variables on the right are referred to as tail
- P(A,B) = P(A|B)P(B) = P(B|A)P(A) =>
  - P(A|B) = P(B|A)P(A) / P(B)
  - This is Bayes theorem
  - Used during inference

Joint probability

- E.g. P(Raining, Windy)
- Sums to 1

Raining	Windy = False	Windy = True
False	0.64	0.16
True	0.1	0.1

Marginalization

P(Raining, Windy)

Raining	Windy = False	Windy = True	Sum
False	0.64	0.16	0.8
True	0.1	0.1	0.2

P(Raining)

Raining = False	Raining = True
0.8	0.2

For discrete variables we sum, whereas for continuous variables we integrate

Marginalization – multiple variables

P(A,B,C,D)

B	C	D	A = True	A = False
True	True	True	0.0036	0.0054
True	True	False	0.0098	0.0252
True	False	True	0.0024	0.0486
True	False	False	0.0042	0.1008
False	True	True	0.0256	0.0864
False	True	False	0.0432	0.1728
False	False	True	0.0064	0.2016
False	False	False	0.0048	0.2592

P(A,C)

C	A = True	A = False
True	0.0822	0.2898
False	0.0178	0.6102

Marginal probability P(A,C) created by marginalizing B and D from the joint probability P(A,B,C,D)

Multiplication

P(B,D | A, C)

B	C	D	A = True	A = False
True	True	True	0.0438	0.0186
True	True	False	0.1192	0.0870
True	False	True	0.1348	0.0796
True	False	False	0.2360	0.1652
False	True	True	0.3114	0.2981
False	True	False	0.5255	0.5963
False	False	True	0.3596	0.3304
False	False	False	0.2697	0.4248

×

P(A,C)

C	A = True	A = False
True	0.0822	0.2898
False	0.0178	0.6102

=

P(A,B,C,D)

B	C	D	A = True	A = False
True	True	True	0.0036	0.0054
True	True	False	0.0098	0.0252
True	False	True	0.0024	0.0486
True	False	False	0.0042	0.1008
False	True	True	0.0256	0.0864
False	True	False	0.0432	0.1728
False	False	True	0.0064	0.2016
False	False	False	0.0048	0.2592

Instantiation – (evidence)

P(A=False,B,C,D)

B	C	D	A = True	A = False
True	True	True	0.0036	0.0
True	True	False	0.0098	0.0
True	False	True	0.0024	0.0
True	False	False	0.0042	0.0
False	True	True	0.0256	0.0
False	True	False	0.0432	0.0
False	False	True	0.0064	0.0
False	False	False	0.0048	0.0

P(B, C, D)

C	D	B=True	B=False
True	True	0.0036	0.0256
True	False	0.0098	0.0432
False	True	0.0024	0.0064
False	False	0.0042	0.0048

Bayesian network inference

Joint probability – Bayesian network

- If we multiply all the distributions of a Bayesian network together, we get the joint distribution over all variables
- What can we do with the joint?
- Any evidence **e** is information we know (e.g. D=True)

$$P(\mathbf{X}, \mathrm{e}) = \sum_{\mathbf{U} \setminus \mathbf{X}} P(\mathbf{U}, \mathrm{e}) = \sum_{\mathbf{U} \setminus \mathbf{X}} \prod_{i} P(\mathbf{U}_i | pa(\mathbf{U}_i))\mathrm{e}$$

U = universe of variables
X = variables being predicted
e= evidence on any variables

Just use the joint over all variables?

- We could perform the same tasks if memory and time were not an issue.
- The problem?
  - Exponential increases in size with discrete variables
- The answer?
  - Bayesian network inference

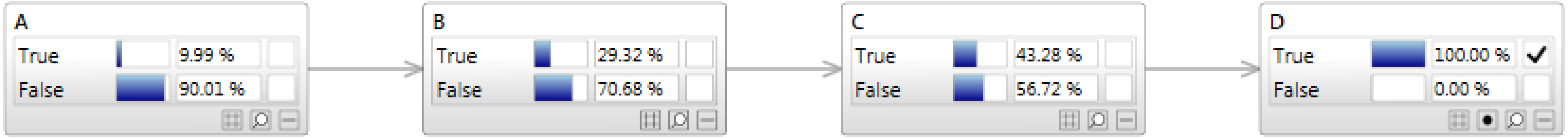
Distributive law

$$if\, A \notin \mathbf{X}, A \in \mathbf{Y}, then \sum_{A} \phi_{\mathbf{X}} \phi_{\mathbf{Y}} = \phi_{\mathbf{X}} \sum_{A} \phi_{\mathbf{Y}}$$

This simply means that if we want to marginalize out the variable A we can perform the calculations on the subset of distributions that contain A

ϕ is a probability distribution over the variables in the subscript

Consider calculating P(A|D=True)



$$P(A|\mathbf{e}) \propto \sum_{B,C,D} P(A)P(B|A)P(C|B)P(D|C)e_D$$

Distributive law

$$P(A|\mathbf{e}) \propto P(A)\sum_{B} P(B|A)\sum_{C} P(C|B)\sum_{D} P(D|C)e_D$$

Elimination order

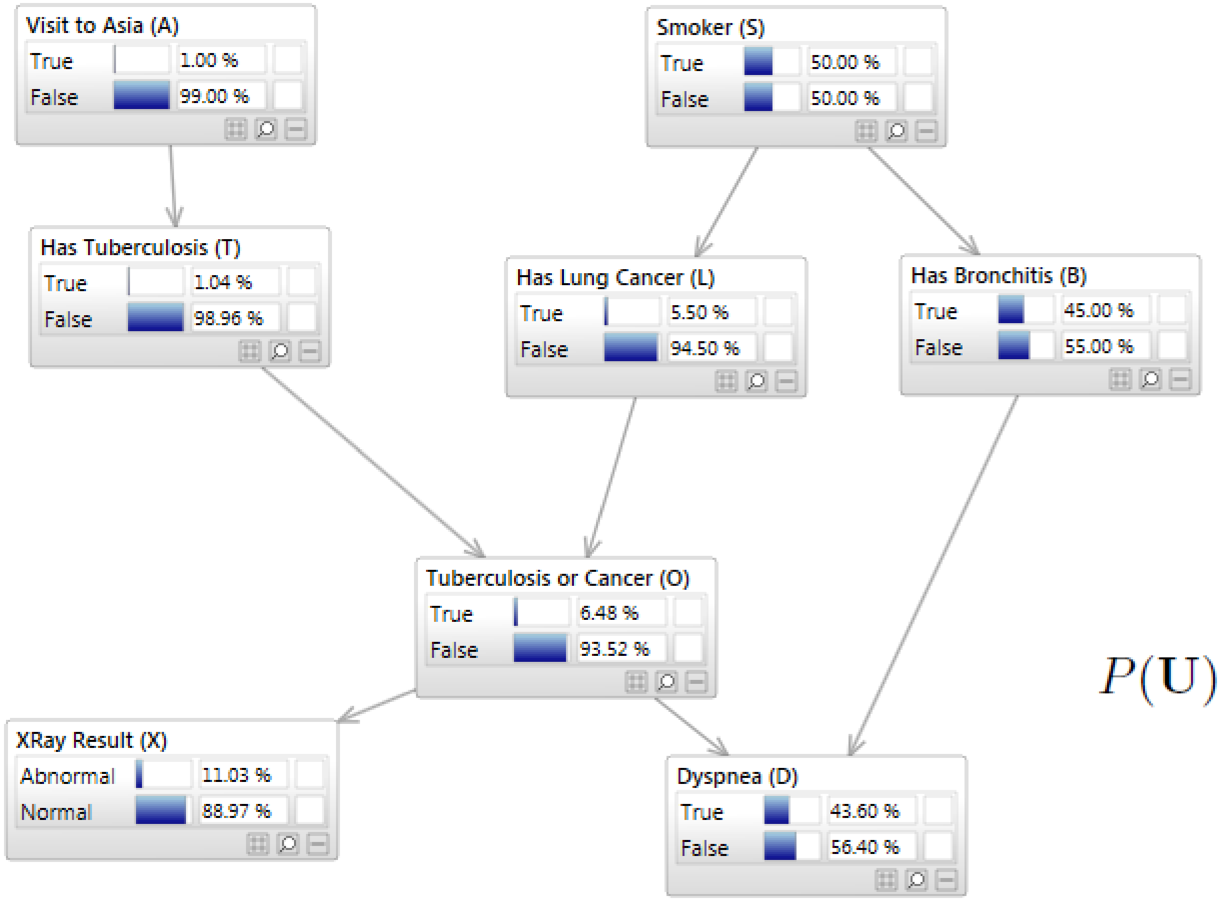
- The order in which marginalization is performed is called an elimination order.
- Many different possible orders
- NP hard
- A number of algorithms exist to determine orderings that work well in practise
  - E.g. pick the variable(s) that result in the smallest distribution to be marginalized at each step
  - Multiple variables can be eliminated at each step.

$$P(A|\mathrm{e}) \propto P(A) \sum_{B} P(B|A) \sum_{C} P(C|B) \sum_{D} P(D|C) e_D$$

Junction trees

- What if we want to predict all variables, not just A?
- We could use the previous procedure known as Variable Elimination multiple times.
- Or we can use a junction tree (join tree)
  - Simply the tree formed by eliminating all variables in the same way as before

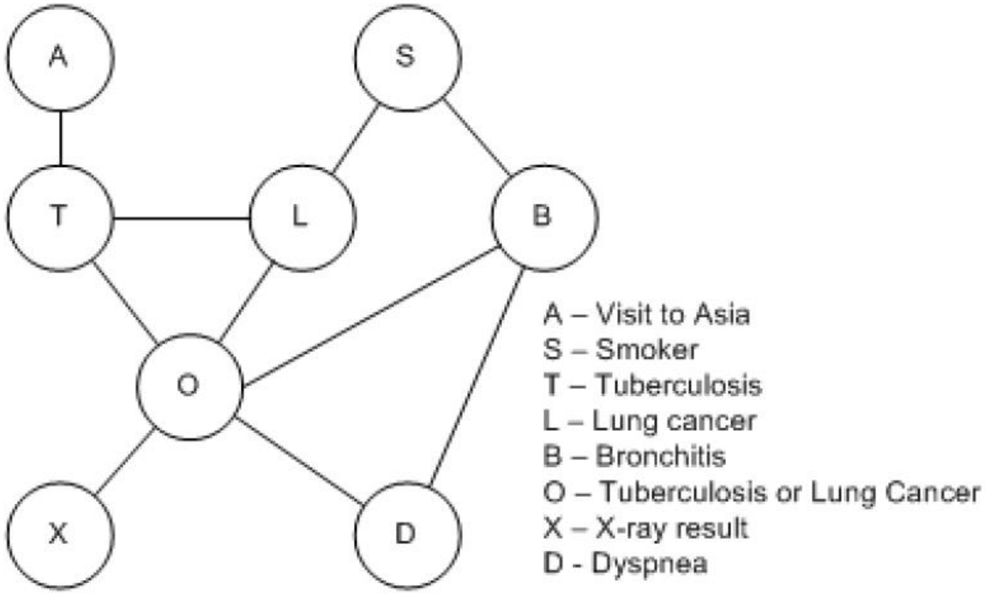
Asia network

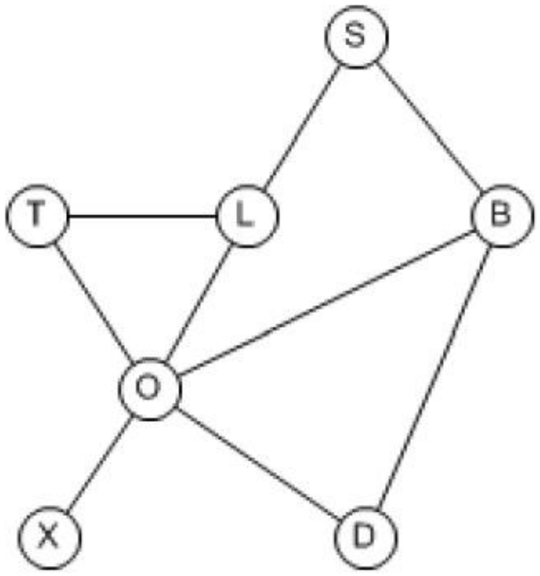


Node	Distribution
A	$P(A)$
T	$P(T\|A)$
S	$P(S)$
L	$P(L\|S)$
B	$P(B\|S)$
O	$P(O\|T,L)$
X	$P(X\|O)$
D	$P(D\|O,B)$

$$P(\mathbf{U}) = P(A)P(S)P(T|A)P(L|S)\times P(B|S)P(O|T,L)(X|O)P(D|B,O)$$

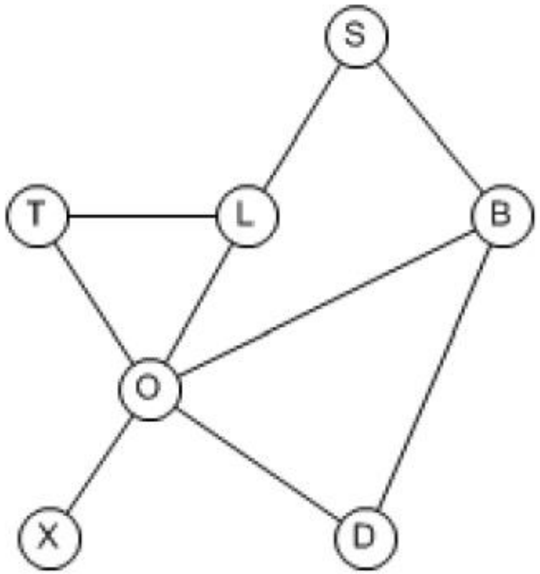
Elimination



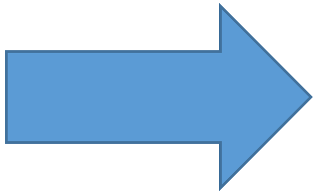


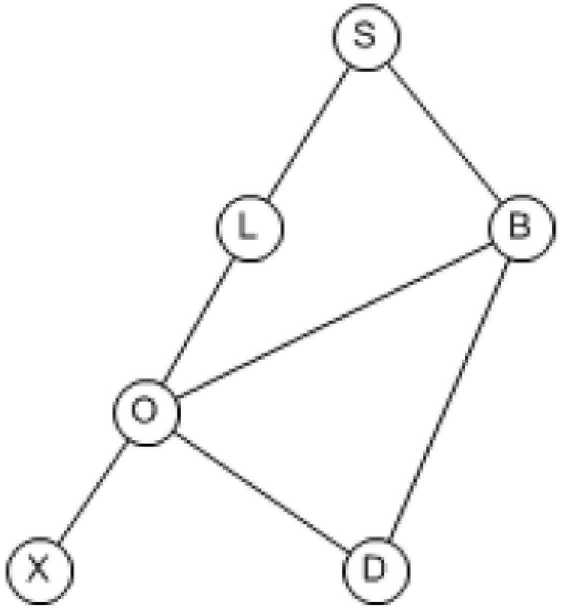
Domain graph after elimination of A

Elimination



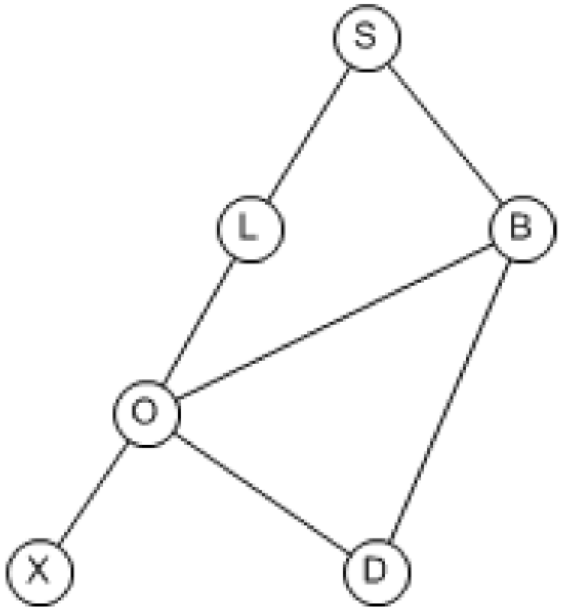
Domain graph after elimination of A



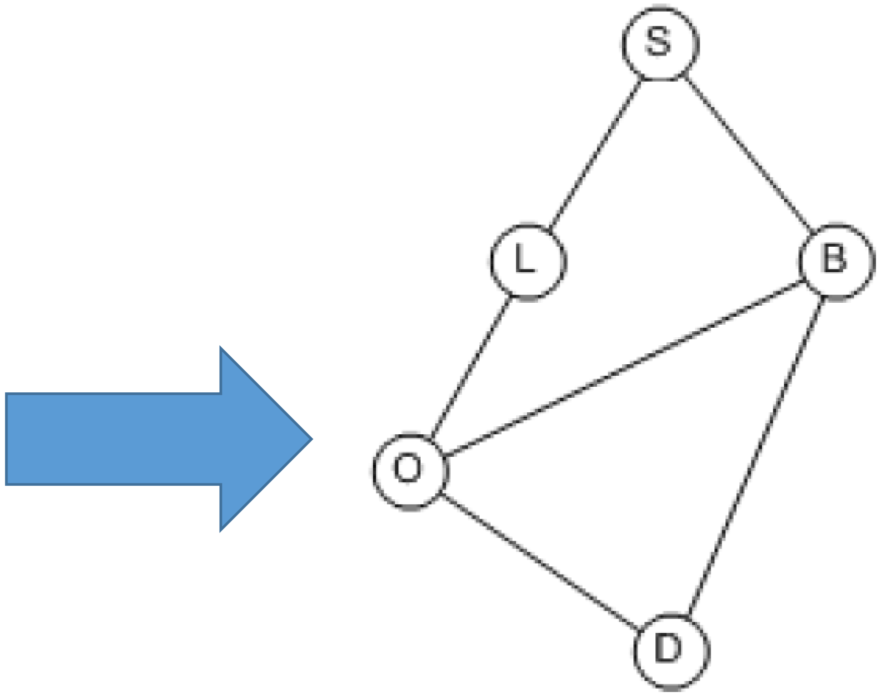


Domain graph after elimination of T

Elimination

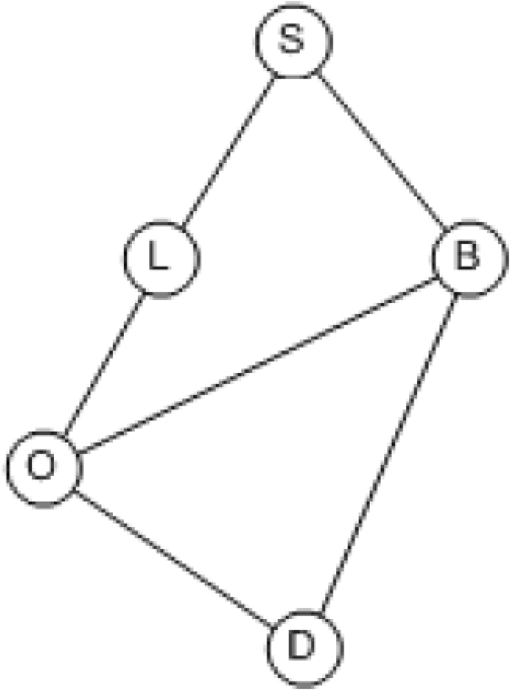


Domain graph after elimination of T

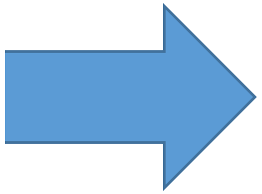


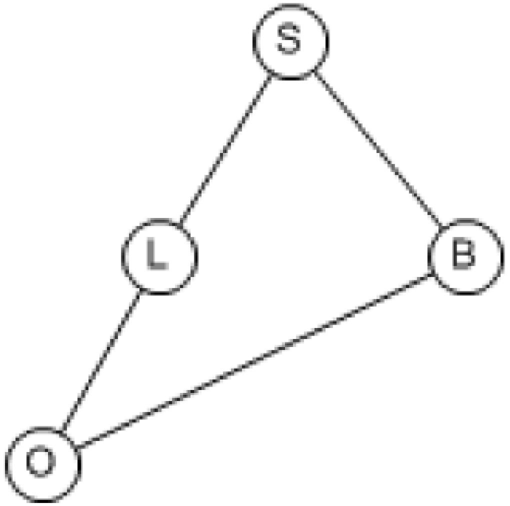
Domain graph after elimination of X

Elimination



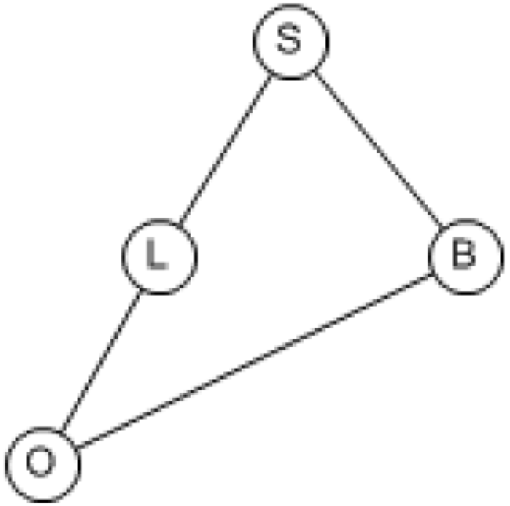
Domain graph after elimination of X



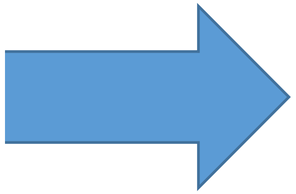


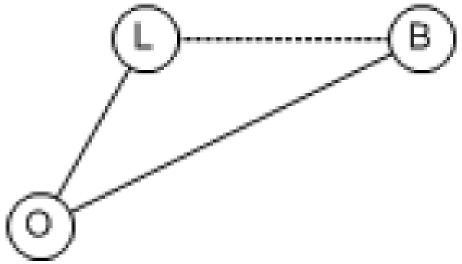
Domain graph after elimination of D

Elimination



Domain graph after elimination of D



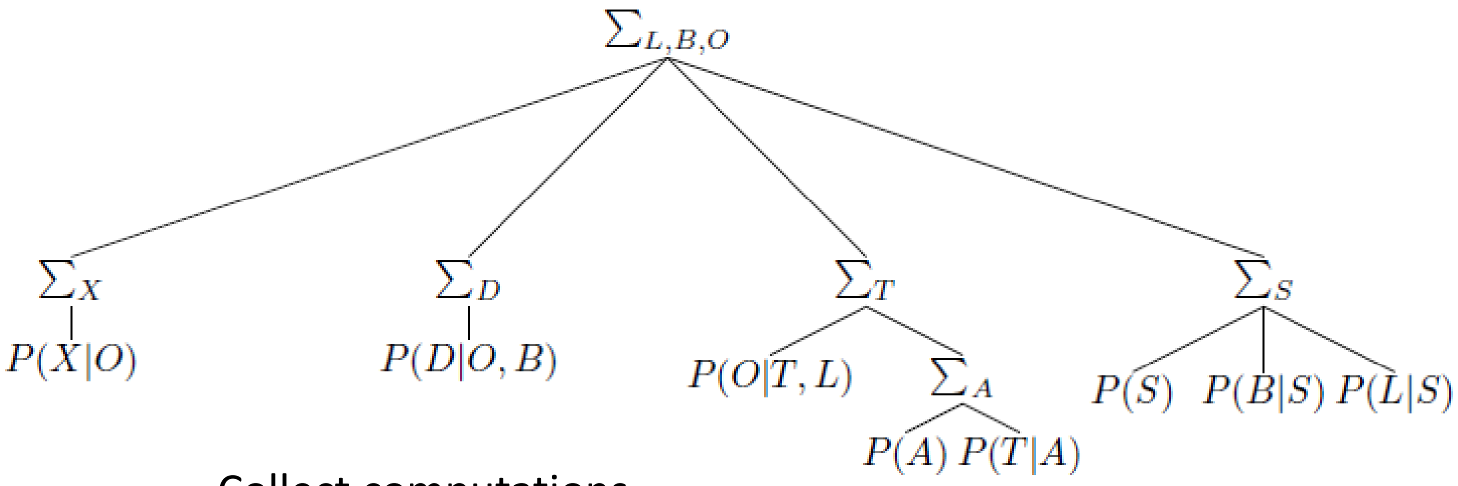


7.: Domain graph after elimination of S. The dotted line is a required fill-in.

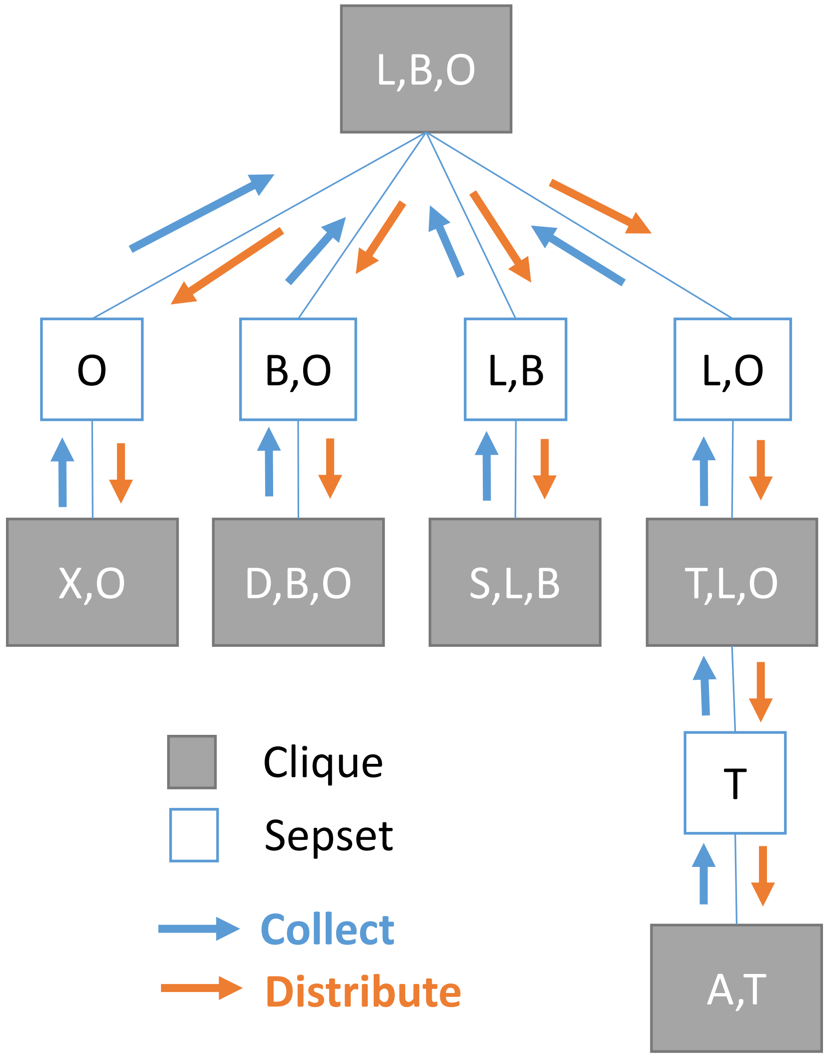
The complete elimination order is…

{A}, {T}, {X}, {D}, {S}, {L,B,O}

Junction trees

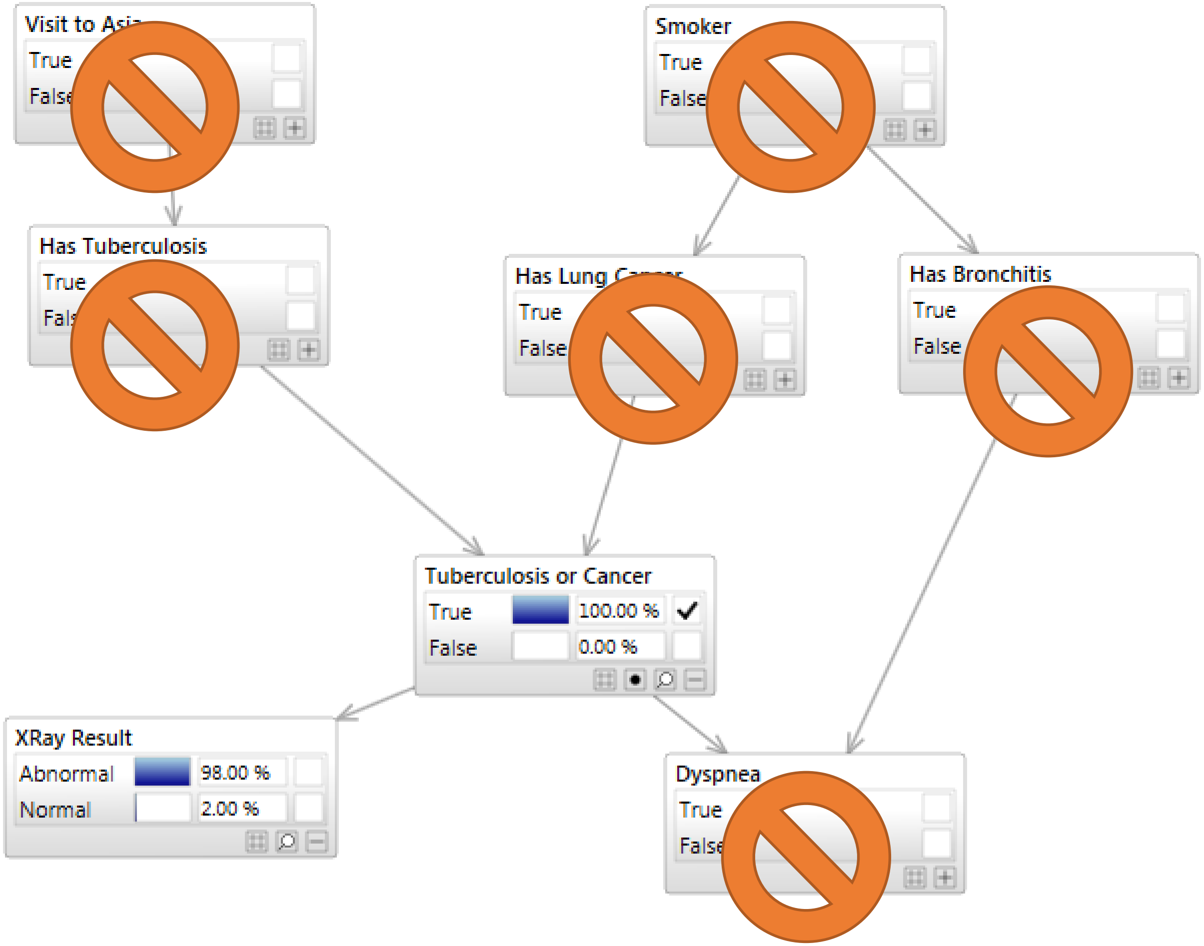


Collect computations



- Collect towards the root {L,B,O} – similar to variable elimination
- Distribute from the root {L,B,O} back to the leaves – allows us to calculate all marginals– P(A), P(X), P(B), P(L) etc…

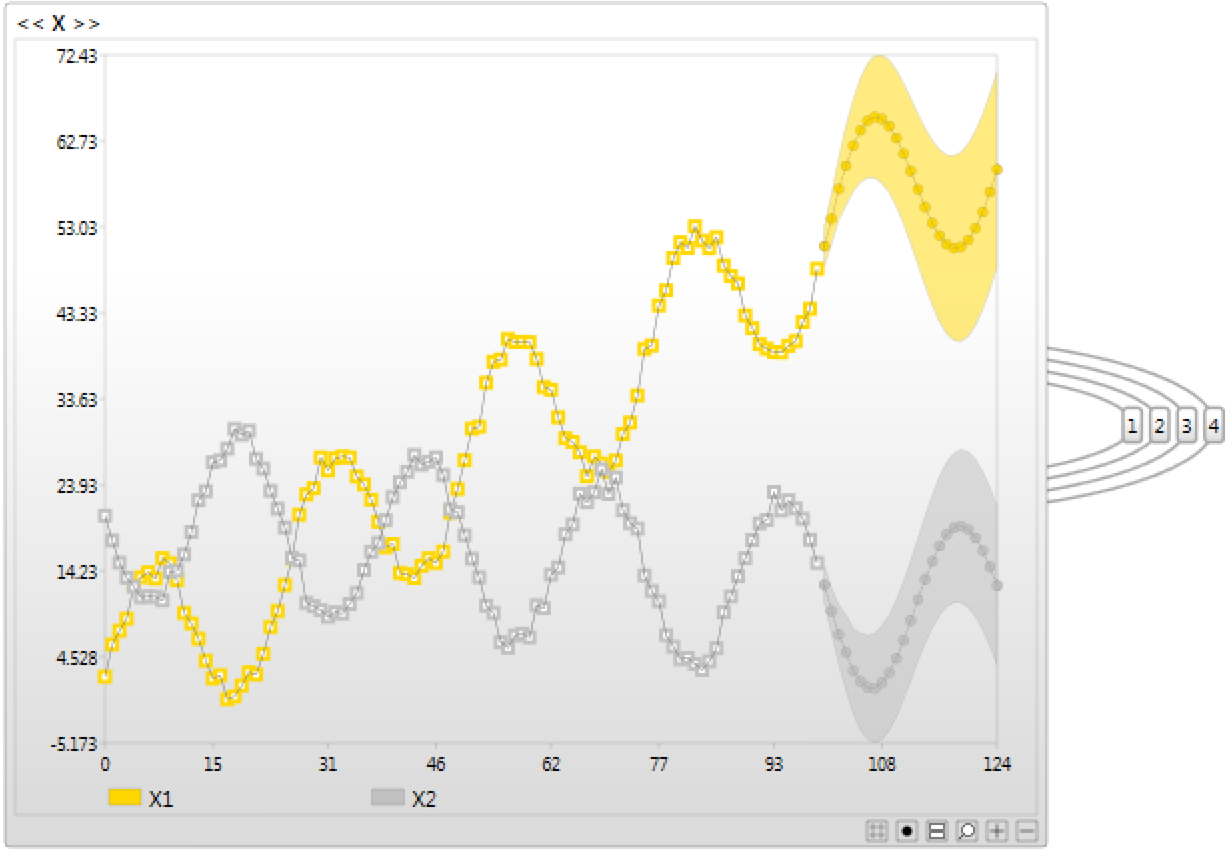
Relevance – Bayes ball algorithm



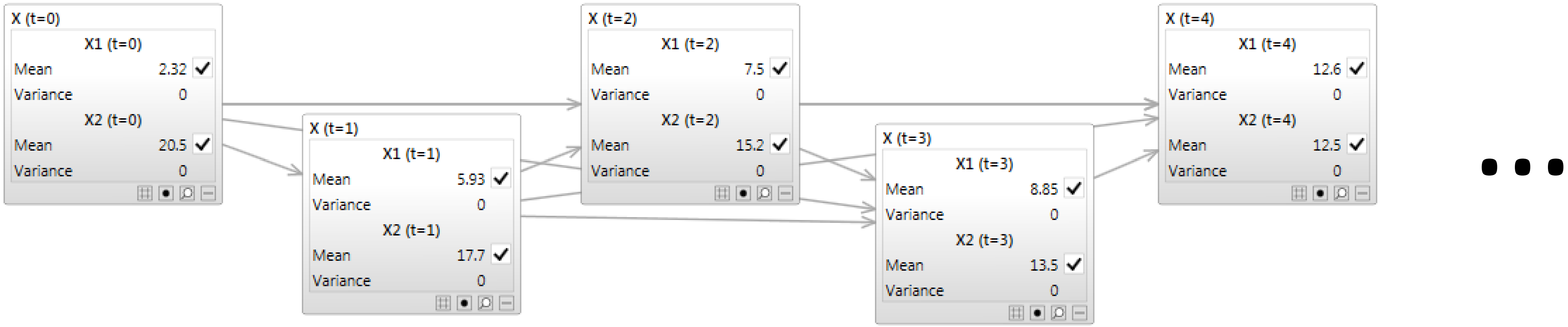
Inference with time series

-Dynamic Bayesian networks

Dynamic Bayesian networks



Unrolling



We could unroll, and use standard methods

Distributions that understand time

$$P(X1[t], X2[t] \mid X1[t-1], X2[t-1])$$

	X1[t]	X2[t]
Intercept	3.076214646583...	-1.58979124120...
Covariance (X1[t])	4.142028922619...	-1.63113437658...
Covariance (X2[t])	-1.63113437658...	2.023002098810...
Weight (X1[t-1])	0.995368300968...	-0.00816950459...
Weight (X2[t-1])	0.026861977953...	0.942548514594...

Note that X1 appears in the same distribution twice, but at different times

Time aware distributions

- Marginalization & multiplication is well defined
- We can use all the existing algorithms
- We can construct queries like…
- P(X1@t=4)
  - Returns probabilities for discrete, mean & variance for continuous
- P(X1@t=4, X2@t=4)
  - Joint time series prediction (funnel)
- P(X1@t=2, X1@t=3)
  - Across different times
- P(A, X1@t=2)
  - Mixed static & temporal
- Log-likelihood of a multivariate time series
  - Anomaly detection

Distributed parameter learning

Different types of scalability

Data size

Big data?

Connectivity

(discrete -> exponential)

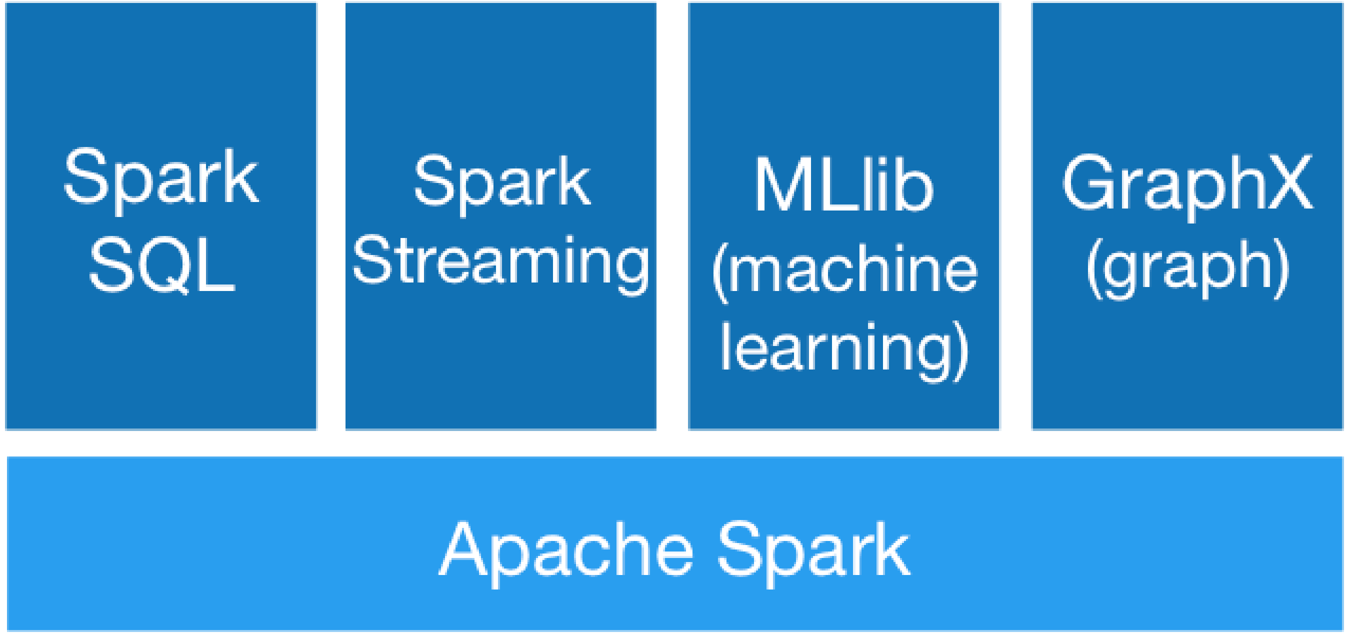
Network size,

Rephil > 1M nodes

Inference

(distributed)

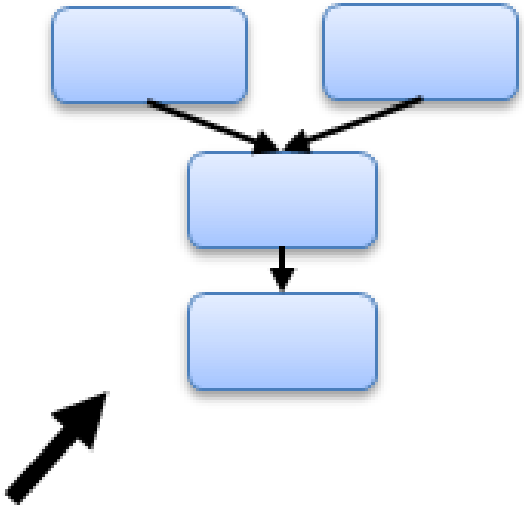
Apache Spark



hadoop

Cassandra

RDD Objects



```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
```

Apache Spark

- RDD (Resilient distributed dataset)
- Can be in memory
- Code automatically converted to DAG execution engine
- Serialization of variables

Distributed architecture

1. Distribute Bayesian network to worker nodes
2. Calculate the sufficient statistics per partition
   - This often requires an inference algorithm per thread/partition
3. Aggregate the sufficient statistics (reduce)
4. Update the Bayesian network based on the new statistics
5. Return to 1 until convergence

Distributed parameter learning

Node 1

X	Y
10.2	12.5
14.5	3.2
15.6	8.2
9.2	12.2
15.8	9.2
4.5	2.1
…	…

∑ stats

X	Y
3.4	3.2
6.6	1.6
5.5	4.3
12.4	8.9
-1.1	-2.4
4.5	4.2
…	…

∑ stats

X	Y
2.0	4.0
8.1	3.4
2.2	7.7
15.1	1.2
4.6	4.5
2.4	1.9
…	…

∑ stats

Node 2

X	Y
10.2	12.5
14.5	3.2
15.6	8.2
9.2	12.2
15.8	9.2
4.5	2.1
…	…

∑ stats

X	Y
3.4	3.2
6.6	1.6
5.5	4.3
12.4	8.9
-1.1	-2.4
4.5	4.2
…	…

∑ stats

X	Y
2.0	4.0
8.1	3.4
2.2	7.7
15.1	1.2
4.6	4.5
2.4	1.9
…	…

∑ stats

Node 3

X	Y
10.2	12.5
14.5	3.2
15.6	8.2
9.2	12.2
15.8	9.2
4.5	2.1
…	…

∑ stats

X	Y
3.4	3.2
6.6	1.6
5.5	4.3
12.4	8.9
-1.1	-2.4
4.5	4.2
…	…

∑ stats

∑ stats ∑ stats ∑ stats ∑ stats
∑ stats ∑ stats ∑ stats ∑ stats

Distributed parameter learning

Example – distributed learning

```
val sc = new SparkContext(conf)

// hard code some test data.  Normally you would read data from your cluster.
val data = createRDD.cache()

// A network could be loaded from a file or stream
// we create it manually here to keep the example self contained
val network = createNetwork

val parameterLearningOptions = new ParameterLearningOptions

// Bayes Server supports multi-threaded learning
// which we want to turn off as Spark takes care of this
parameterLearningOptions.setMaximumConcurrency(1)

/// parameterLearningOptions.setMaximumIterations(...) // this can be useful to limit the number of iterations

val config = new MemoryNameValues // we could also use broadcast variables

val output = ParameterLearning.learnDistributed(network, parameterLearningOptions,
  new BayesSparkDistributer[Seq[(Double, Double)]](
    data,
    config,
    (ctx, iterator) => new TimeSeriesEvidenceReader(ctx.getNetwork, iterator)
  ))
```

Distributed time series prediction

```
// make some time series predictions into the future

val predictions = Prediction.predict[TimeSeries](
  network,
  testData,
  Seq(
    PredictVariable("X1", Some(PredictTime(5, Absolute))), PredictVariance("X1", Some(PredictTime(5, Absolute))),
    PredictVariable("X2", Some(PredictTime(5, Absolute))), PredictVariance("X2", Some(PredictTime(5, Absolute))),
    PredictVariable("X1", Some(PredictTime(6, Absolute))), PredictVariance("X1", Some(PredictTime(6, Absolute))),
    PredictVariable("X2", Some(PredictTime(6, Absolute))), PredictVariance("X2", Some(PredictTime(6, Absolute))),
    PredictLogLikelihood() // this value can be used for Time Series anomaly detection
  ),
  (network, iterator) => new TimeSeriesReader(network, iterator))

predictions.foreach(println)
```

Thank you