Tutorial on Optimal Algorithms for Learning Bayesian Networks

James Cussens, Brandon Malone, Changhe Yuan

Monday, August 5th, afternoon
https://sites.google.com/site/ijcai2013bns/

About presenters

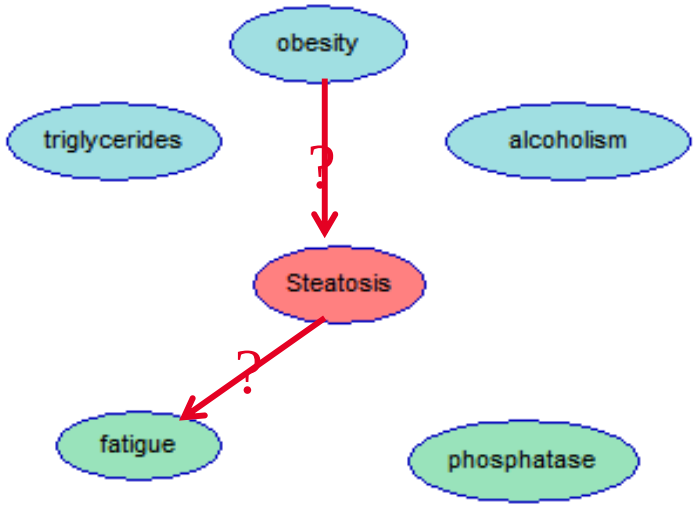
- **Dr. James Cussens**
  - Senior Lecturer in the Dept of Computer Science at the University of York, UK
- **Dr. Brandon Malone**
  - Postdoctoral researcher at the University of Helsinki in the Complex Systems Computation Group
- **Dr. Changhe Yuan**
  - Associate Professor of Computer Science at Queens College/City University of New York
  - Director of the Uncertainty Reasoning Laboratory (URL Lab).

Outline

- **Basics of Bayesian networks (30 minutes)**
- **Scoring functions (30 minutes)**
- **Dynamic programming (30 minutes)**
- **Admissible heuristic search (45 minutes)**
- **Integer linear programming (60 minutes)**
- **Empirical evaluations (15 minutes)**

Understanding uncertain relations

- Many research problems involve understanding the uncertain relations between a set of random variables
  - Ignorance: Limited knowledge
  - Laziness: Details ignored



- Idea: Using a joint probability distribution to capture the relations

Basic probability theory

- **Probability theory has to do with experiments that have a set of distinct outcomes/values**
  - **E.g. toss a coin with outcome being head or tail**
- **A random variable is a function on an outcome space: it assigns a unique value to each outcome in the domain**
  - *Weather:* sunny, rainy, cloudy, snowy
- **In the case of a finite domain, every subset of the outcome space is called an event**
  - e.g., *Weather* = (*sunny* $\vee$ rainy)
- **A discrete random variable has an associated probability distribution**
  - *P(Weather) = <sunny: 0.72, rainy: 0.1, cloudy: 0.08,snowy: 0.1>*

Some probability concepts

- **Prior or unconditional probabilities of events**

  e.g., *P(Earthquake = true) = 0.1, P(Weather = sunny) = 0.72*

- **Joint probability distribution (multivariate distribution) for a set of random variables gives the probability of every atomic event on those random variables**

  P(Weather, Alarm) = a 4 × 2 matrix of values:

Weather =	*sunny*	*rainy*	*cloudy*	*snowy*
Earthquake = true	*0.072*	*0.01*	*0.008*	*0.01*
Earthquake = false	*0.648*	*0.09*	*0.072*	*0.09*

- **Conditional or posterior probabilities**

  P(Earthquake | Alarm):

Alarm =	*yes*	*no*
Earthquake = true	*0.22*	*0.03*
Earthquake = false	*0.78*	*0.97*

Some probability rules

- **Conditional probability rule:**
  $P(a \mid b) = P(a \wedge b) / P(b)$ if $P(b) > 0$

- **Product rule (chain rule):**
  $P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$
  $P(a \wedge b \wedge c) = P(a \mid b,c)\, P(b|c)\, P(c)$

- **Total probability rule: For any event φ, sum the atomic events where φ is true:** $P(\varphi) = \Sigma_{\omega:\omega \models \varphi} P(\omega)$
  $P(X=x) = \Sigma\_Y\, P(X=x, Y)$

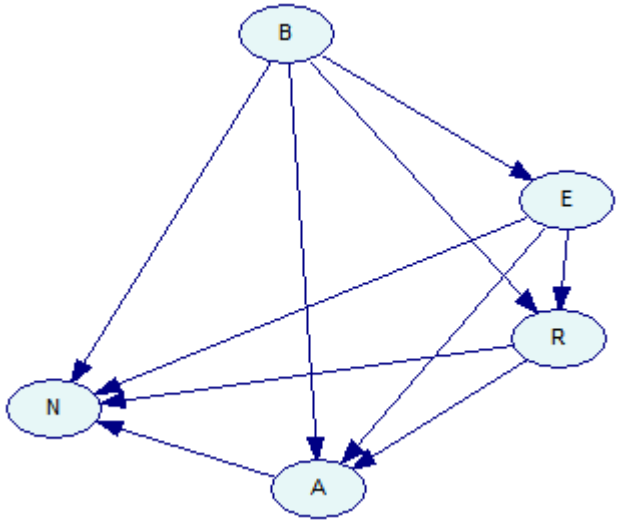
- **Bayes rule:**
  $P(a \mid b) = P(b \mid a)\, P(a) / P(b)$
  $P(Cause|Effect) = P(Effect|Cause)\, P(Cause) / P(Effect)$

Graphical representation of probability distribution

Representation: a joint probability distribution $P(B, E, R, A, N)$ over five binary variables

$$P(B,E,R,A,N) = P(B)P(E \mid B)P(R \mid B,E)P(A \mid B,E,R)P(N \mid B,E,R,A)$$

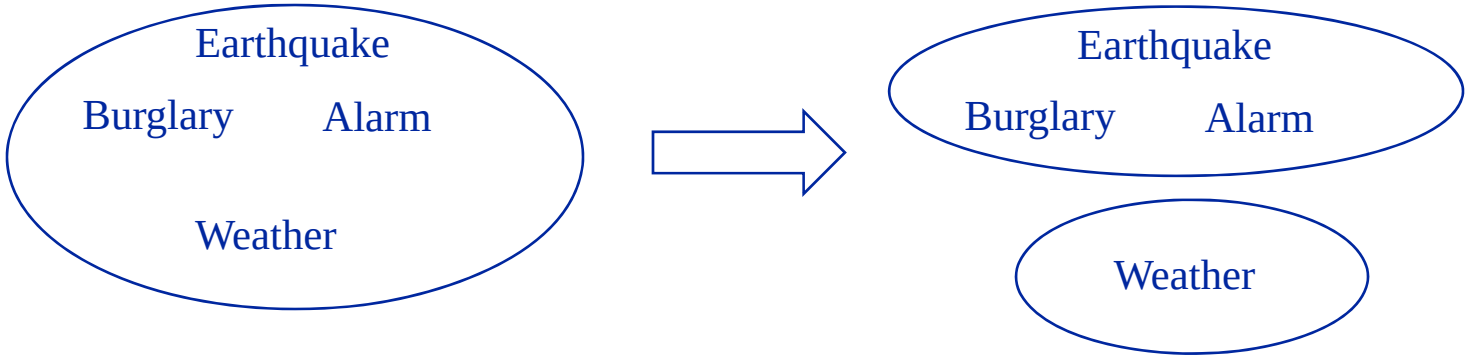
- **How many free parameters (probabilities) are needed?**
  - **5-dimensional table: 31**
  - **Graphical representation: 31**



Independence

- ***A* and *B* are independent iff**

  $P(A|B) = P(A)$ or $P(B|A) = P(B)$ or $P(A, B) = P(A)\,P(B)$



P(Earthquake, Burglary, Alarm, Weather)
= P(Earthquake, Burglary, Alarm) P(Weather)

- **31 free parameters reduced to 10**

Conditional independence

- **If there is an *earthquake*, the probability that *alarm* sounds doesn't depend on *radio* announcement**
- ***Alarm* is conditionally independent of *RadioAnnounce* given Earthquake**

  P(Alarm | RadioAnnounce , Earthquake) = P(Alarm | Earthquake) or

  P(Alarm, RadioAnnounce | Earthquake) = P(Alarm | Earthquake) P(RadioAnnounce | Earthquake)

Graphical representation of probability distribution

Representation: a joint probability distribution over five binary variables

$P(Burglary, Earthquake, RadioAnnounce, Alarm, NeighborCall)$

given the following conditional independence relations:

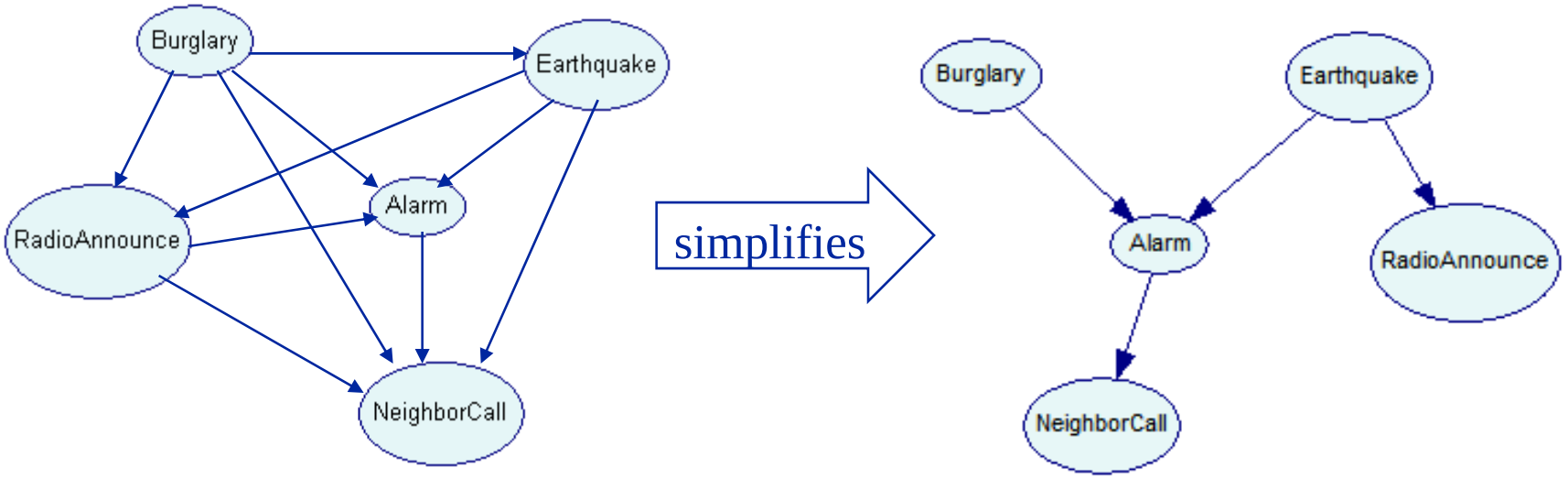
$B \perp E$

$R \perp N,A,B \mid E$

$N \perp B,E,R \mid A$

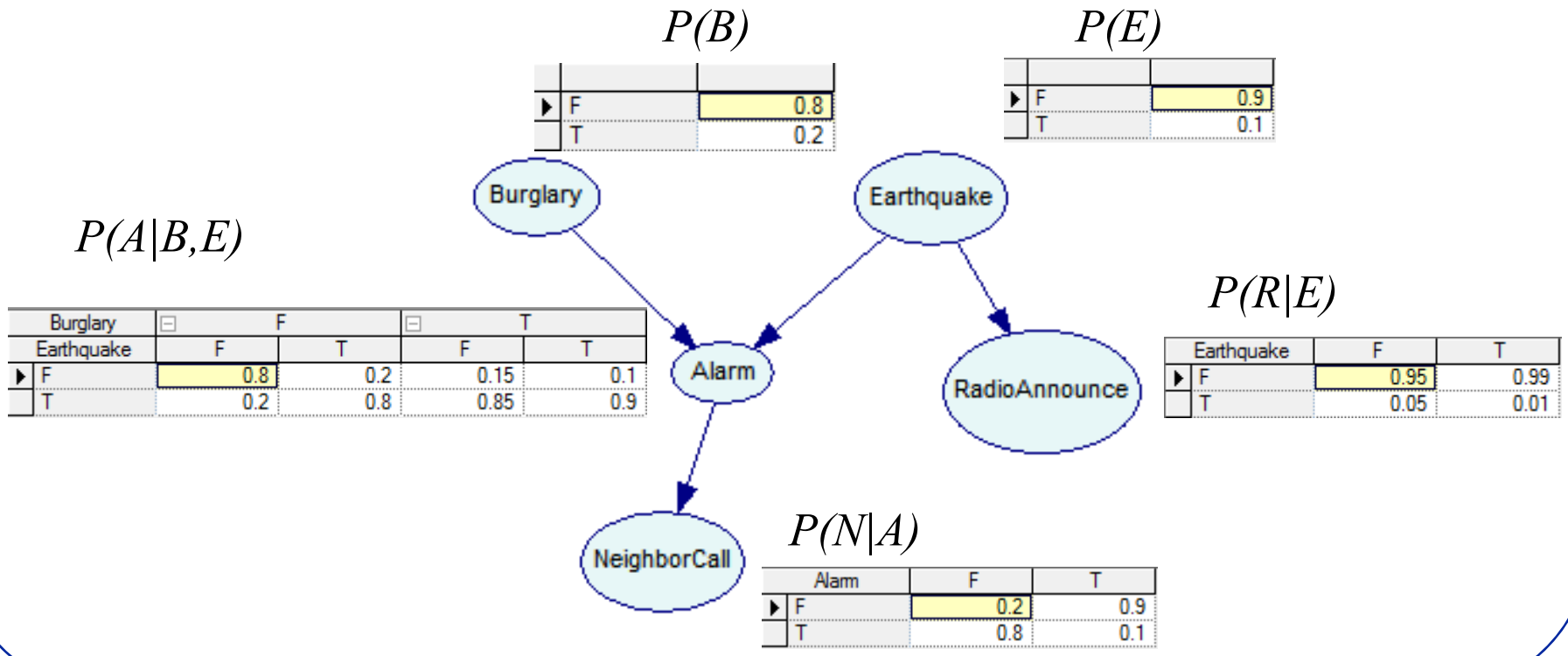
$$P(B,E,R,A,N) = P(B)P(E)P(A|B,E)P(N|A)P(R|E)$$

31 free parameters reduced to 10 !

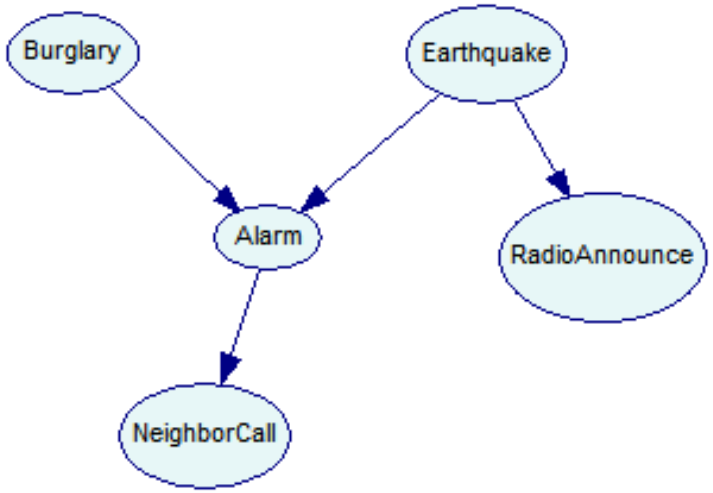


Bayesian networks

- **A Bayesian Network is a directed acyclic graph (DAG) in which:**
  - **A set of random variables makes up the nodes in the network.**
  - **A set of directed links or arrows connects pairs of nodes.**
  - **Each node has a conditional probability table that *quantifies* the effects the parents have on the node.**



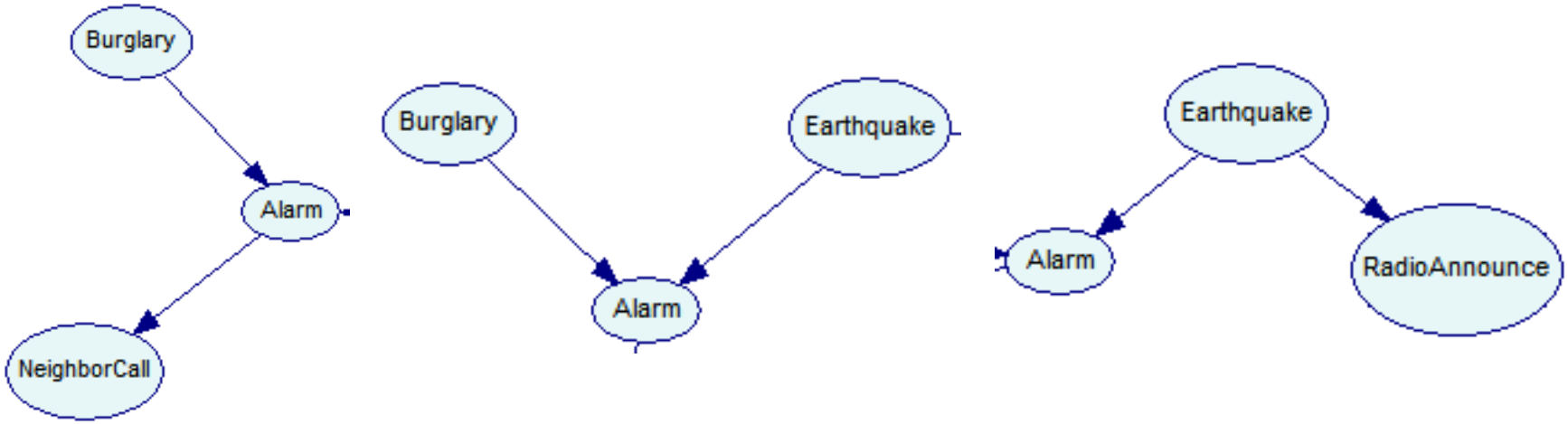
Chain rule



$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i))$$

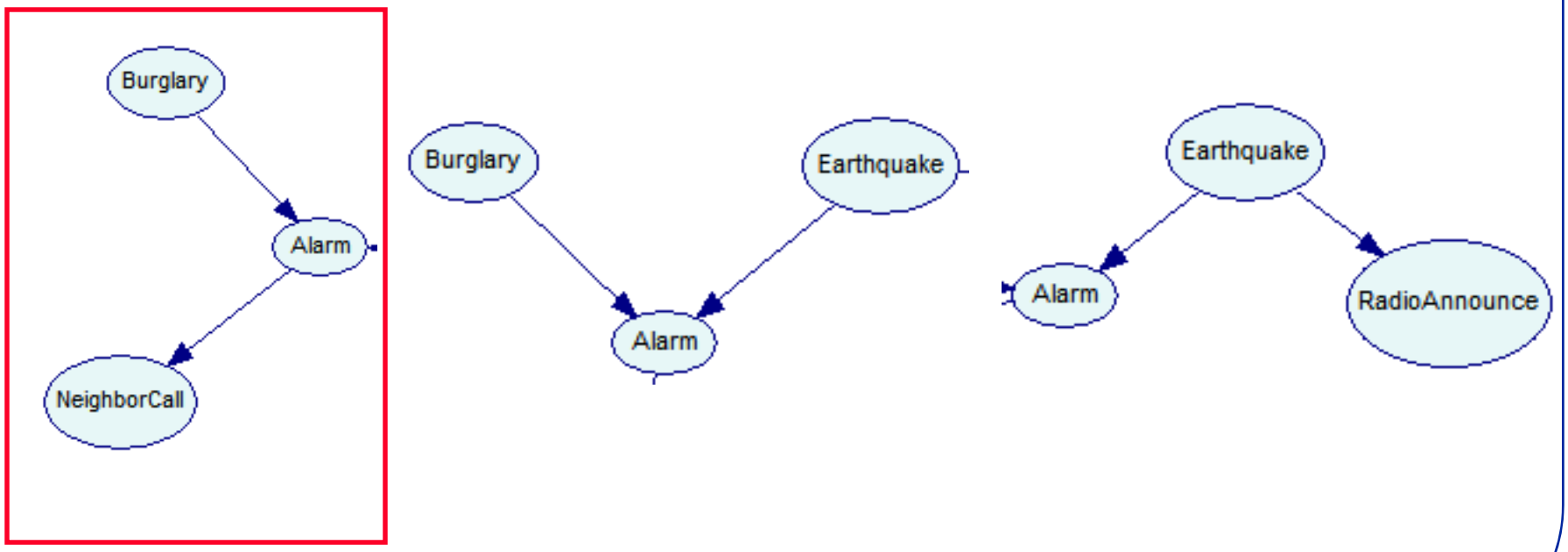
Independences in BNs

- **Three basic independence structures as building blocks:**



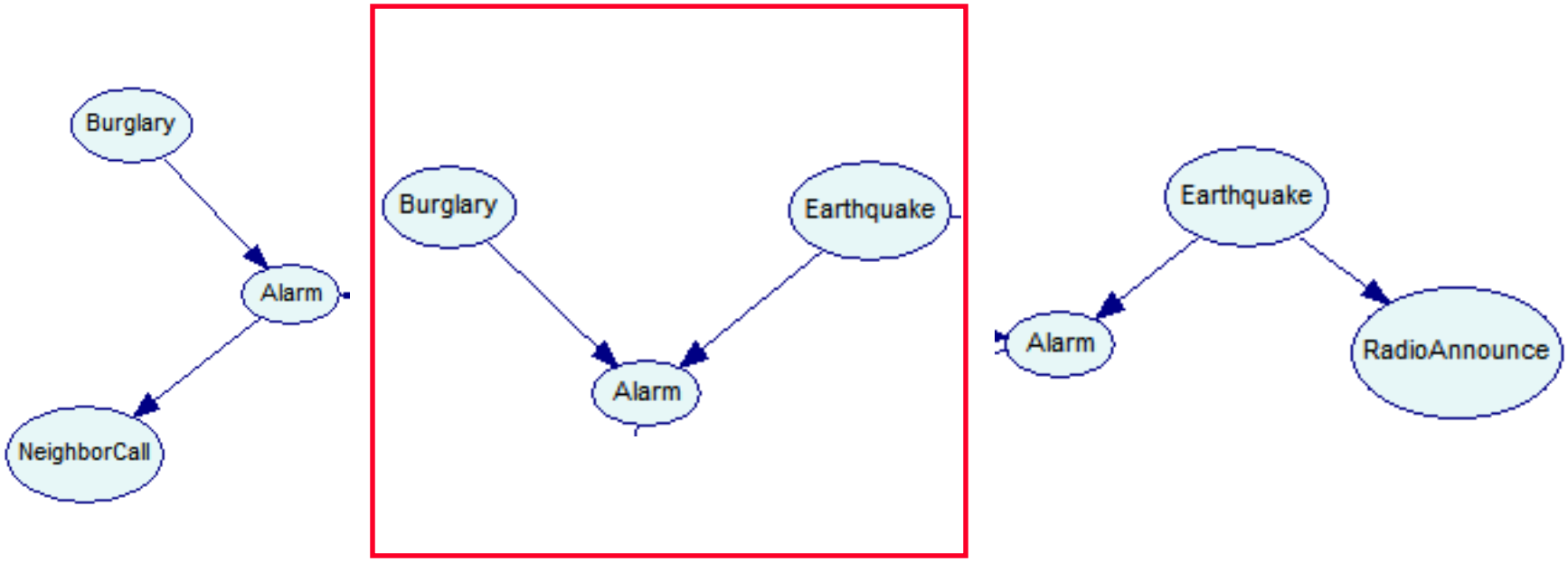
Independences in BNs

- **Indirect cause**: ***Burglary*** **is independent of** ***NeighborCall*** **given** ***Alarm***
  - *P(N|A, B) = P(N|A)*
  - *P(N, B|A) = P(N|A)P(B|A)*



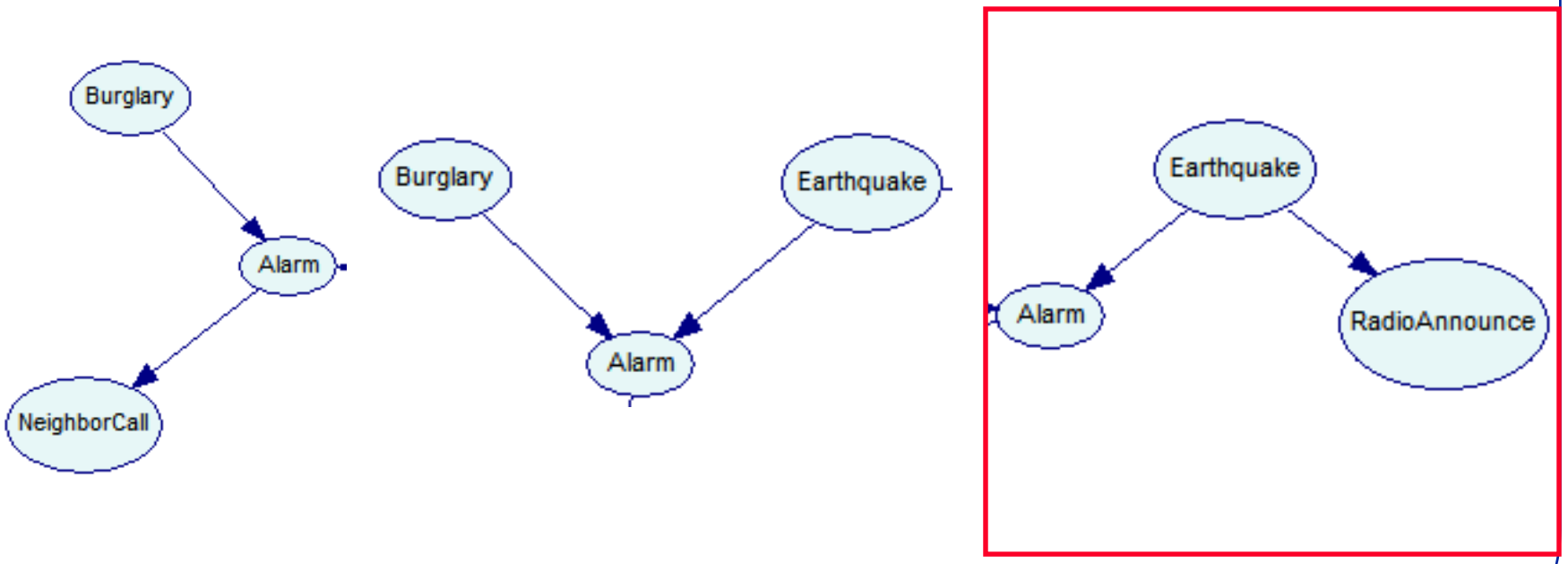
Independences in BNs

- **Common effect**:
  - ***Burglary*** **is independent of** ***Earthquake*** **when Alarm is not known**
  - ***Burglary*** **and** ***Earthquake*** **become dependent given** ***Alarm*****!!!**



Independences in BNs

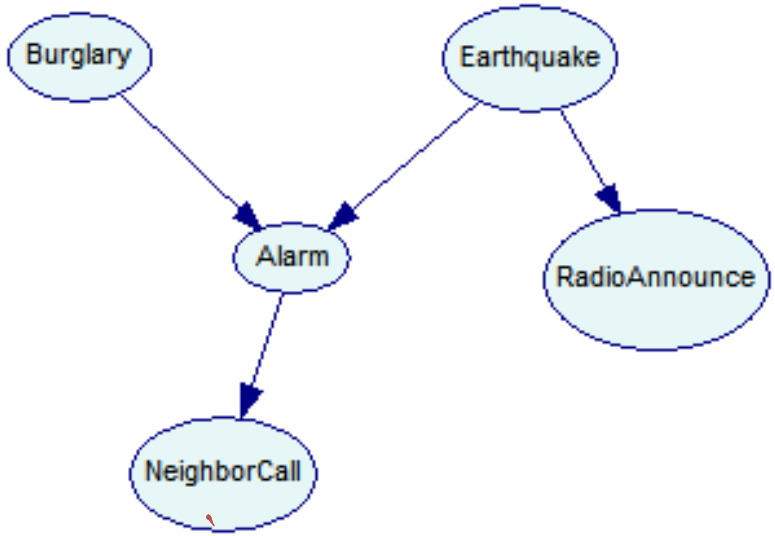
- **Common cause**: ***Alarm* is independent of *RadioAnnounce* given *Earthquake***
  - $P(A|E, R) = P(A|E)$
  - $P(A, R|E) = P(A|E)P(R|E)$



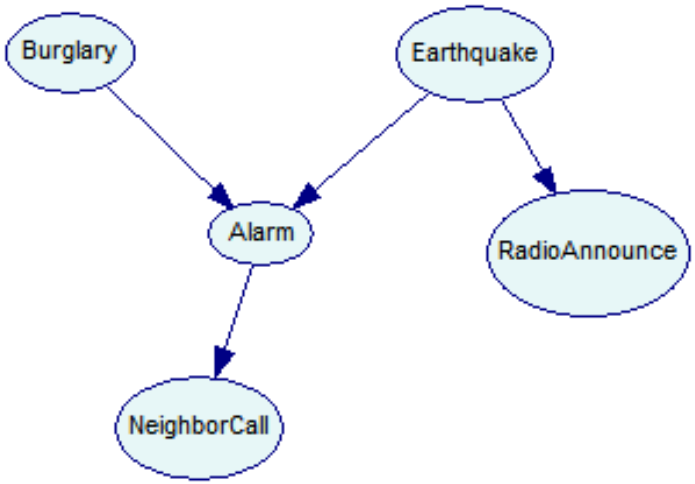
Markov assumption

- **Each variable is independent from its non-descendants, given its parents in a Bayesian network**

- $N \perp B, E, R \mid A$
- $R \perp B, A, N \mid E$
- ...



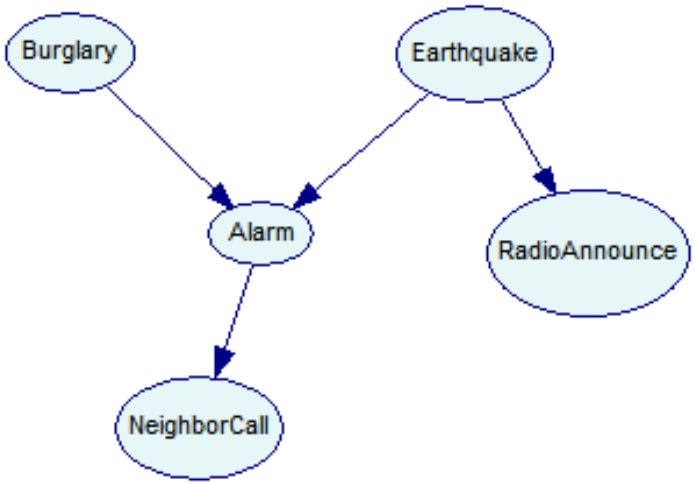
Inference tasks (1)



- **Belief updating: computing the posterior probabilities of an unobserved variable given observed variables**

$$P(Burglary \mid RadioAnnounce = true, NeighborCall = true)$$

Inference tasks (2)



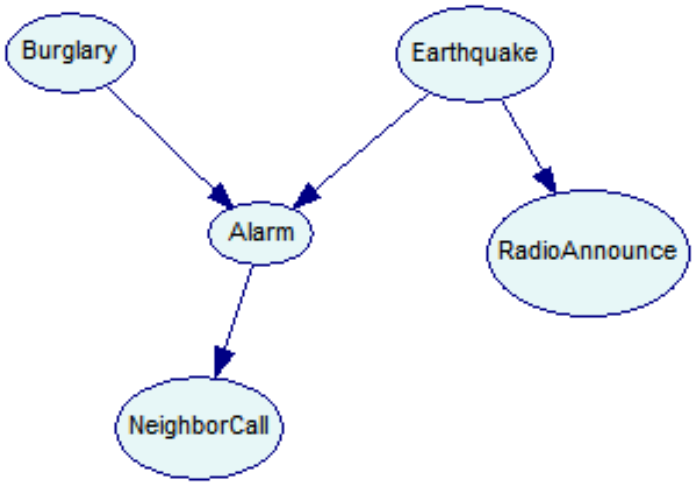
- **Most probable explanation (MPE) queries: compute the most likely joint state of all unobserved variables**

$$\arg\max_{Burglary, Earthquake, Alarm} P(Burglary, Earthquake, Alarm \mid RadioAnnounce = true, NeighborCall = true)$$

- **Maximum A Posteriori Assignment (MAP) queries: compute the most likely joint state of some unobserved variables**

$$\arg\max_{Burglary, Earthquake, Alarm} P(Burglary, Earthquake \mid RadioAnnounce = true, NeighborCall = true)$$

Inference tasks (3)



- **Most Relevant Explanation (MRE) queries: compute the most relevant partial explanation for the observed evidence**

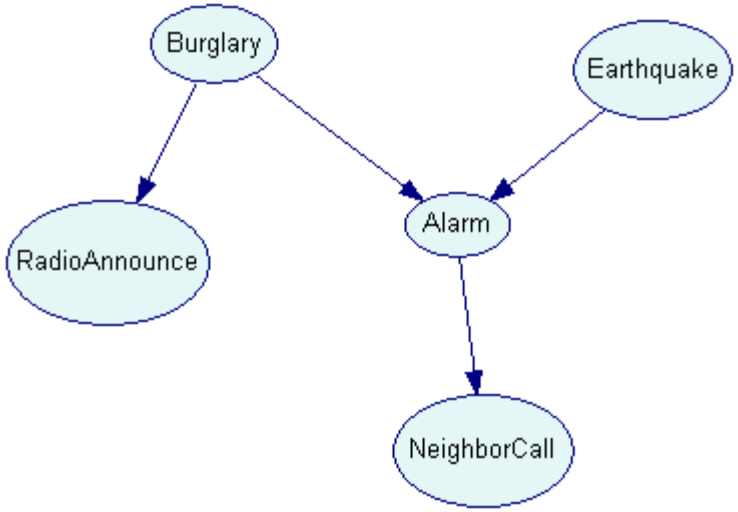
$$\arg\max_{Burglary, Earthquake} GBF(Burglary, Earthquake \mid RadioAnnounce = true, NeighborCall = true)$$

[Yuan, Lim, Lu, JAIR-11]

Efficient inference

Efficient algorithms for inference and learning computation by exploring the graph structure

Suppose we want to compute *P(R)*?



#sum=15

$$P(R) = \sum_E \sum_B \sum_A \sum_N P(B,E,R,A,N)$$

$$= \sum_E \sum_B \sum_A \sum_N P(B)P(E)P(R \mid B)P(A \mid B,E)P(N \mid A)$$

$$= \sum_B P(B)P(R \mid B) \sum_E P(E) \sum_A P(A \mid B,E) \sum_N P(N \mid A)$$

$$= \sum_B P(B)P(R \mid B)$$

#sum=1, #multiply=2

Advantages of Bayesian networks

- Provide intuitive and compact graphical representations of the uncertain relations between the random variables
- Provide well-understood principles for
  - Incorporating prior knowledge with observational data
  - Handing missing data
  - Knowledge discovery from data
- Able to solve both prediction and explanation tasks

  (Unlike many other machine learning methods that are mainly predictive methods)

Applications of Bayesian networks

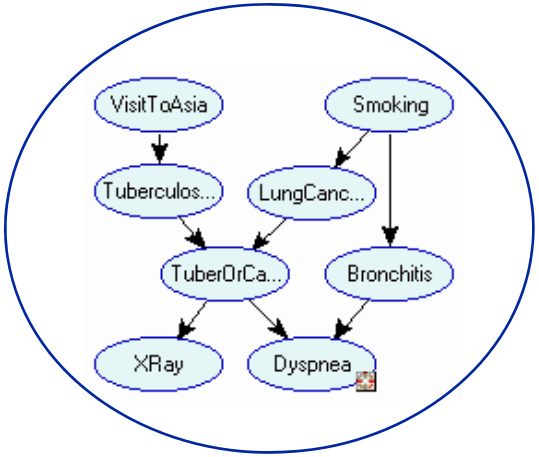
- Robotics
- Bioinformatics
- Security
- User modeling
- Vision
- Data mining
- Medical and machine diagnosis/prognosis
- Information retrieval
- Planning
- Natural language interpretation
- Fraud detection
- Planning
- …

Learning Bayesian networks

- Very often we have data sets
- We can learn Bayesian networks from these data

100	100	100	90	390	97.5%
100	95	100	80	375	93.8%
100	100	100	90	390	97.5%
80	95	100	90	365	91.3%
100	100	100	100	400	100.0%
100	100	100	100	400	100.0%
90	95	100	90	375	93.8%
90	95	100	90	375	93.8%
100	100	100	90	390	97.5%
100	100	100	100	400	100.0%
100	90	100	90	380	95.0%
95	90	100	80	365	91.3%
100	95	100	80	375	93.8%
100	95	100	80	375	93.8%
100	100	100	100	400	100.0%

data



structure

Success	0.2
Failure	0.8

	Success	Failure
Good	0.4	0.1
Moderate	0.4	0.3
Poor	0.2	0.6

numerical parameters

Goals of structure learning

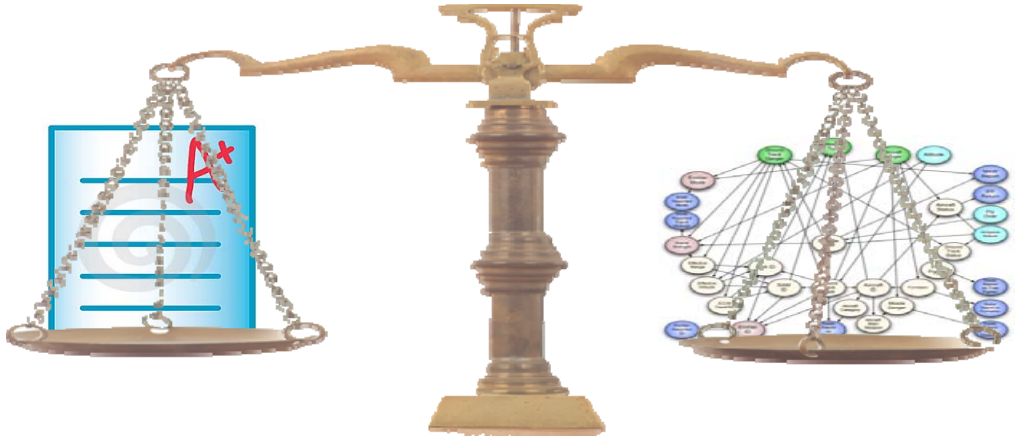
- **Knowledge discovery**
  - learn the dependency structure relating variables
- **Density estimation**
  - estimate a statistical model of the underlying distribution

Major learning approaches

- **Score-based** structure learning
  - Find the highest-scoring network structure
- **Constraint-based** structure learning
  - Find a network that best explains the dependencies and independencies in the data
- **Hybrid** approaches
  - Integrate constraint- and/or score-based structure learning
- Bayesian model **averaging**
  - Average the prediction of all possible structures

Score-based learning

- Find a Bayesian network that optimizes a given scoring function



- Two major research issues evident from above
  - How to define a scoring function?
  - How to formulate and solve the optimization problem?

Contents of this tutorial

- **Exact algorithms for score-based structure learning**
  - **Scoring functions**
  - **Dynamic programming**
  - **Admissible heuristic search**
  - **Integer linear programming**
  - **Empirical evaluations**

Outline

- Basics of Bayesian networks (30 minutes)
- ➢ Scoring functions (30 minutes)
- Dynamic programming (30 minutes)
- Admissible heuristic search (45 minutes)
- Integer linear programming (60 minutes)
- Evaluation (15 minutes)

Scoring Functions

- Bayesian Dirichlet Family (BD)
- Minimum Description Length (MDL)
- Factorized Normalized Maximum Likelihood (fNML)
- Score Pruning
- Practicalities

Assumptions (Generally)

1. Multinomial sample.
2. Complete data. Datasets are complete.
3. Parameter independence.
   - Global. Parameters of each variable are independent.
   - Local. Parent parameters of a variable are independent.
4. Parameter modularity. Parents determine parameters.
5. Dirichlet*. Parameter densities are Dirichlet.
6. Structure possibility**. All complete structures are possible.
7. Likelihood equivalence***. Score of equivalent networks are equal.

[Heckerman 1995]

Bayesian Dirichlet (BD) Scores

Suppose we'd like to find the most likely joint probability of structure and data given prior knowledge.

$$G^* = \underset{G}{\operatorname{argmax}}\, p(D, G|\xi)$$

The probability of a particular structure is

$$p(D, G|\xi) = p(G|\xi) \cdot p(D|G, \xi)$$

We have two components:

- **Evaluation of the structure**
- **Evaluation of the data given the structure**

[Heckerman 1995]

In general, ξ is given as Dirichlet hyperparameters, α.

BD, Evaluation of $p(D|G,\xi)$

First, calculate the probability of each sample

$$p(C_l|D_l,G,\xi) = \prod_{i,j,k} E(\theta_{ijk}|D_l,G,\xi)^{1_{lijk}}$$

Based on the Dirichlet distribution, write this as

$$p(C_l|D_l,G,\xi) = \prod_{i,j,k} \left(\frac{\alpha_{ijk}+N_{ijk}}{\alpha_{ij}+N_{ij}}\right)^{1_{lijk}} \qquad \alpha_{ij} = \sum_k \alpha_{ijk}$$

We can then write the likelihood by multiplying the probability of all samples together as

$$p(D|G,\xi) = \prod_{i,j,k,l} \left(\frac{\alpha_{ijk}+N_{ijk}}{\alpha_{ij}+N_{ij}}\right)^{1_{lijk}}$$

[Heckerman 1995]

Score Probability, $p(D, G|\xi)$

Recall, by chain rule, we have

$$p(D, G|\xi) = p(G|\xi) \cdot p(D|G, \xi)$$

From the previous derivation, we can rewrite this as

$$p(D, G|\xi) = p(G|\xi) \prod_{i,j,k,l} \left(\frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}} \right)^{1_{lijk}}$$

By combining like terms, we can simplify this as

$$p(D, G|\xi) = p(G|\xi) \prod_{i,j} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})}$$

This is the BD scoring function.

If we say that $\alpha_{ijk} = 1$, then we have the K2 metric.

[Heckerman 1995, Cooper and Herskovits 1992]

Some Equivalences

If G_1 and G_2 are Markov equivalent…

- **Prior probabilities and equivalence**

$$p(G_1|\xi) = p(G_2|\xi)$$

- **Likelihood probabilities and equivalence**

$$p(D|G_1,\xi) = p(D|G_2,\xi)$$

- **Score probabilities and equivalence**

$$p(D,G_1|\xi) = p(D,G_2|\xi)$$

[Heckerman 1995]

The BDe and BDeu Scoring Functions

The BD metric does not guarantee likelihood equivalence.

We can restrict the hyperparameters to ensure likelihood equivalence and derive the BDe metric.

$$\alpha_{ijk} = \alpha \cdot p(x_i = k, PA(i) = j | G, \xi)$$

We can also use uninformative hyperparameters. This gives the BDeu metric.

$$\alpha_{ijk} = \frac{\alpha}{r_i \cdot q_i}$$

[Heckerman 1995]

Prior Structures, $p(G|\xi)$

Many methods can be used to specify prior probabilities over structures.

Heckerman et al., propose solicitation of a prior structure from experts and the following function.

Suppose *G* and the prior structure differ by δ arcs.

$$p(G|\xi) = c\kappa^{\delta}$$

This does not satisfy prior equivalence in general.

A uniform prior is often assumed.

[Heckerman 1995]

The BDeu Scoring Function

Putting everything together...

$$p(D,G|\xi) = p(G|\xi)\prod_{i}^{n}\prod_{j}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$$

If we assume uniform priors and log space...

$$p(D,G|\xi) = \sum_{i}^{n}\sum_{j}^{q_i}\log\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})} + \sum_{k}^{r_i}\log\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$$

Limitations of BDeu

- **(And all other methods) Parameter independence is violated when data is missing.**
- **(And all other methods) Experimental data is different than observational data.**
- **Problem: How do I specify priors over structures?**
- **Problem: How do I specify α?**

Minimum Description Length (MDL)

MDL views *learning* as *data compression*.

Traditionally, MDL consists of two components:

- **Model encoding**
- **Data encoding, using the model**

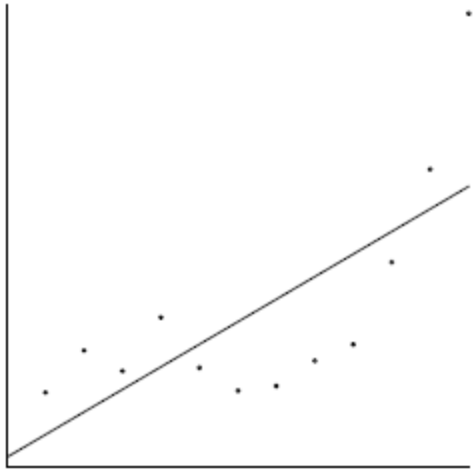
A few properties:

- **Formalizes Occam's Razor**
- **Works regardless of 'true' model**

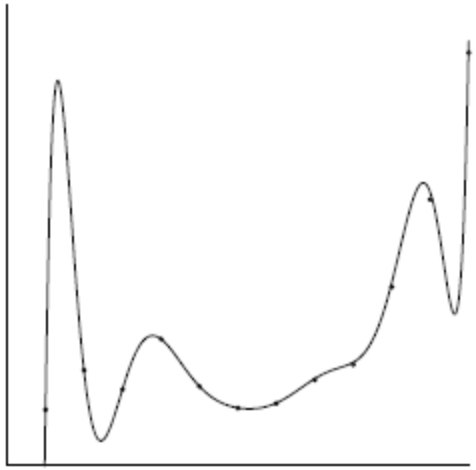
Note: In the BN literature, "MDL" refers to the two-part version of MDL.

[Rissanen 1978, Grunwald 2005]

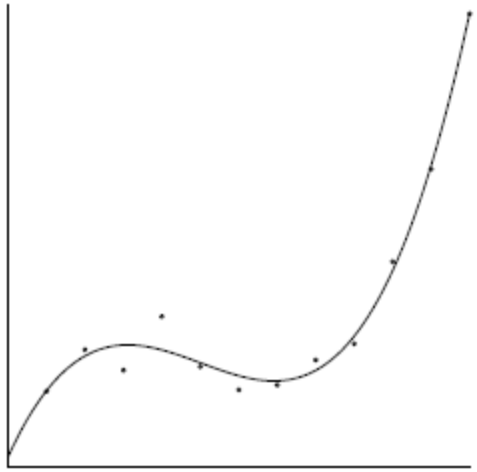
Simple MDL Example



Short model encoding
Long data encoding



Long model encoding
Short data encoding



Medium model encoding
Medium data encoding

[Grunwald 2005]

Encoding Bayesian Network Models

We must encode:

- **Parents of each node**

  We need $\log(n)$ bits per parent.

- **Parameters of conditional probability distributions***

  We need $(r_i\text{-}1)*q_i$ parameters for x_i.

  We need to use $\frac{\log N}{2}$ bits per parameters to satisfy certain minimax criteria.

The total complexity is $\sum_{i}^{n} \log n * |PA_i| + \frac{\log N}{2}(r_i - 1)q_i$ **.**

Other choices for encodings are possible.

*** Assuming we use conditional probability tables [Lam and Bacchus 1994, Suzuki 1993]**

Encoding Data Given a Bayesian Network Model, 1

Each instantiation i is assigned a unique binary string (codeword), with length $\approx -\log p_i$

So the length to encode the entire dataset is $\approx -N \sum_i p_i \log p_i$

Problems:

- Unknown probabilities p_i
- Exponentially many unique instantiations

Observations:

- Cross-entropy gives an upper bound on length.
- The BN factorizes.

[Lam and Bacchus 1994]

Encoding Data Given a Bayesian Network Model, 2

Solution: Approximate cross-entropy using the sufficient statistics from the data.

The length to encode the data for X_i with the model is

$$-\sum_{j}^{q_i}\sum_{k}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}}$$

[Lam and Bacchus 1994]

MDL as a Scoring Function for Bayesian Networks

As derived here, the entire MDL score for a network is:

$$-\sum_{i}^{n}\sum_{j}^{q_i}\sum_{k}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \log n * |PA_i| - \frac{\log N}{2}(r_i - 1)q_i$$

Another commonly used form does not include the cost to record the parents and is written as:

$$-\sum_{i}^{n}\sum_{j}^{q_i}\sum_{k}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{\log N}{2}q_i = \sum_{i}^{n} -LL(X_i|PA_i) + \frac{\log N}{2}(r_i - 1)q_i$$

Problem: which network encoding is ”best?”

Normalized Maximum Likelihood (NML)

Like MDL, we still view *learning* as *compression*.

NML searches for a *universal code*:

- **Performs well on the data we have**
- **Also performs well on all other datasets (*regret*)**

Formally, $\frac{P(\boldsymbol{D}|M)}{\sum_{\boldsymbol{D}'} P(\boldsymbol{D}'|M)}$ **Maximum likelihood**
Normalized over all datasets

or $\log P(\boldsymbol{D}|M) - \log \sum_{\boldsymbol{D}'} P(\boldsymbol{D}'|M)$

Model parameters are maximized for each D'.

We no longer have to specify an encoding.

[Grunwald 2005]

NML for One-variable Multinomial Sample

Given: $\boldsymbol{D} = \{x_1, \ldots, x_N\}$ **, where each** x_i **is a sample from a single, discrete random variable with** K **values.**

Then the NML distribution for that dataset is

$$\frac{\prod_k^K (N_k/N)^{N_k}}{C(K,N)}$$

$$C(K,N) = \sum_{N'_1+\cdots+N'_K=N} \frac{N!}{N'_1! \ldots N'_K!} \prod_k^K (N'_k/N)^{N'_k}$$

Where C(K, N) is the regret term and the sum is over all datasets of size N.

C(K, N) can be computed in linear time. **[Kontkanen 2007]**

NML for Bayesian Networks

For a particular structure, M: $\mathrm{LL}(\boldsymbol{D}|M) - \log \sum_{\boldsymbol{D}'} P(\boldsymbol{D}'|M)$

Problem: We have not real hope of calculating regret.

Solution: Approximate it with a factorized set of multinomial samples.

For each variable **The regret of those records**

$$\sum_{i}^{n} \sum_{j}^{q_i} C(r_i, N_{ij})$$

For each parent instantiation

[Silander et al. 2008]

Factorized NML for Bayesian Networks

The factorized NML score for Bayesian networks is

$$\sum_{i}^{n}\sum_{j}^{q_i}\sum_{k}^{r_i} -N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - C(r_i, N_{ij})$$

Note that we do not pick the encoding or any other parameters.

We also did not pick hyperparameters.

Decomposability

Consider all of the scoring functions we discussed.

BDeu	$\sum_i^n \sum_j^{q_i} \log \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})} + \sum_k^{r_i} \log \frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$
MDL	$\sum_i^n -LL(X_i \mid PA_i) + \frac{\log N}{2}(r_i - 1)q_i$
fNML	$\sum_i^n \sum_j^{q_i} \sum_k^{r_i} -N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - C(r_i, N_{ij})$

All of these are expressed as a sum over the individual variables.

This property is called *decomposability* and will be quite important for structure learning. **[Heckerman 1995, etc.]**

Score Pruning

Observation: All of the scoring functions we have discussed are *decomposable*, of the form

$$Score(G) = \sum_{i}^{n} Score(X_i|PA_i)$$

Theorem: Say $PA_i \subset PA'_i$ and $Score(X_i|PA_i) > Score(X|PA'_i)$. Then PA'_i is not optimal for X_i.

Proof Sketch: Consider structures which differ only by PA_i and PA'_i. We could just use PA_i and not add cycles.

Problem: We still need $Score(X_i|PA_i)$ and $Score(X|PA'_i)$.

[Teyssier and Koller 2005]

Exponential Score Pruning, MDL

The difference between local MDL scores for parents PA and PA' for X_i can be expressed as

$$s_i(PA_i') - s_i(PA_i) = \mathrm{LL}(X_i|PA_i') - \mathrm{K}(X_i|PA_i') - \mathrm{LL}(X_i|PA_i) + \mathrm{K}(X_i|PA_i)$$

$$s_i(PA_i') - s_i(PA_i) \leq -\mathrm{K}(X_i|PA_i') - \mathrm{LL}(X_i|PA_i) + \mathrm{K}(X_i|PA_i)$$

Theorem: Say $PA_i \subset PA'_i$ and the (second) difference is negative. Then neither PA'_i nor any of its supersets are not optimal for X_i.

Proof intuition: LL is always non-positive and K is monotonically increasing. So the penalty must increase more than the LL can decrease.

[de Campos and Ji 2011, Tian 2000]

Exponential Score Pruning, BDeu

We can write BDeu as…

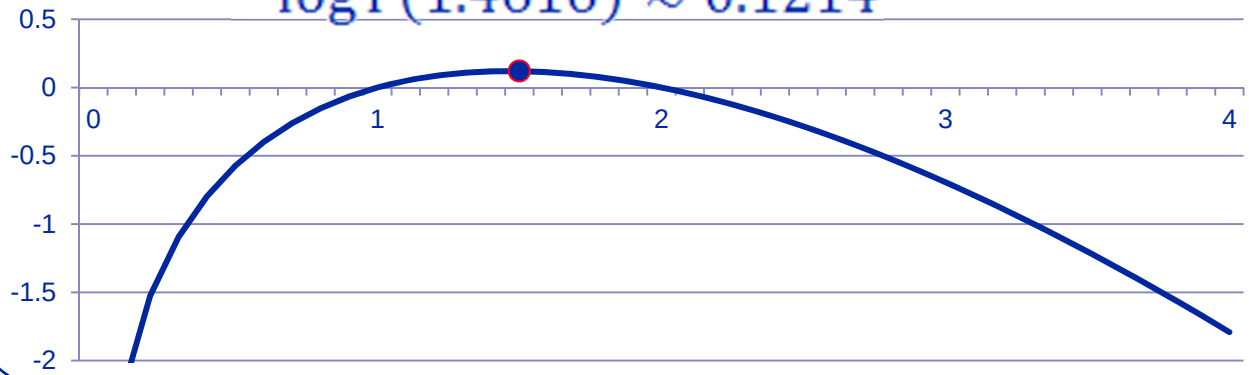
$$s_i(PA(i)) = \sum_j f\left(K_{ij}, (\alpha_{ijk})_{\forall k}\right) + g\left((n_{ijk})_{\forall k}, (\alpha_{ijk})_{\forall k}\right)$$

$$f\left(K_{ij}, (\alpha_{ijk})_{\forall k}\right) = \log\Gamma(\alpha_{ij}) - \sum_k \log\Gamma(\alpha_{ijk})$$

$$g\left((n_{ijk})_{\forall k}, (\alpha_{ijk})_{\forall k}\right) = -\log\Gamma(\alpha_{ij} + N_{ij}) + \sum_k \log\Gamma(\alpha_{ijk} + N_{ijk})$$

Note that $-\log\Gamma(x)$ looks like

$$-\log\Gamma(1.4616) \approx 0.1214$$



[de Campos and Ji 2011]

Exponential Score Pruning, BDeu

Observations: We can bound both f and g based on the behavior of $-\log \Gamma$.

Theorem: Say $PA_i \subset PA'_i$ and $\alpha_{ij}^{PA'_i} \leq 0.8349$ for every j, and $s_i(PA(i)) > -\left|K_{ij}^{PA'_i}\right| \log r_i$. Then neither PA'_i nor any of its supersets are not optimal for X_i.

Proof intuition: Alphas can never increase, and the g term (analogous to LL) can only improve so much.

[de Campos and Ji 2011]

Empirical Results of Score Pruning

	ESS	adult	breast	car	letter	lung	mush	nurse	wdbc	zoo
Max.	0.1	2.1(4)	1.0(1)	0.7(1)	4.5(5)	0.1(2)	4.1(5)	1.2(3)	1.3(2)	1.4(3)
Number	1	2.4(4)	1.0(1)	1.0(2)	5.2(6)	0.4(2)	4.4(7)	1.7(3)	1.7(3)	1.9(4)
of Parents	10	3.3(5)	1.0(1)	1.9(2)	5.9(6)	3.0(4)	4.8(8)	2.1(3)	3.1(4)	3.4(4)
	BIC	2.8(5)	1.0(1)	1.3(2)	6.3(7)	2.1(3)	4.1(4)	1.8(3)	2.7(3)	2.8(3)
Worst-case		14.0	9.0	6.0	16.0	6.0*	22.0	8.0	8.0*	16.0
Final Size	0.1	$2^{4.2}$	$2^{1.5}$	$2^{1.1}$	$2^{8.2}$	$2^{0.2}$	$2^{8.5}$	$2^{1.9}$	$2^{3.6}$	$2^{3.3}$
of the	1	$2^{4.8}$	$2^{1.9}$	$2^{1.6}$	$2^{9.0}$	$2^{0.8}$	$2^{8.9}$	$2^{2.4}$	$2^{4.9}$	$2^{4.4}$
Cache	10	$2^{6.3}$	$2^{3.3}$	$2^{3.0}$	$2^{10.5}$	$2^{10.7}$	$2^{9.8}$	$2^{3.5}$	$2^{12.1}$	$2^{8.9}$
	BIC	$2^{9.3}$	$2^{4.7}$	$2^{4.5}$	$2^{15.3}$	$2^{11.5}$	$2^{13.0}$	$2^{5.6}$	$2^{12.9}$	$2^{10.9}$
Worst-case		$2^{17.9}$	$2^{12.3}$	$2^{8.8}$	$2^{20.1}$	$2^{31.1*}$	$2^{26.5}$	$2^{11.2}$	$2^{28.4*}$	$2^{20.1}$
Implied	0.1	$2^{54.1}$	$2^{13.3}$	$2^{6.3}$	$2^{129.0}$	$2^{8.2}$	$2^{175.7}$	$2^{11.6}$	$2^{90.3}$	$2^{39.3}$
Search	1	$2^{62.1}$	$2^{17.1}$	$2^{8.3}$	$2^{144.8}$	$2^{33.1}$	$2^{186.0}$	$2^{15.4}$	$2^{132.7}$	$2^{60.3}$
Space	10	$2^{91.6}$	$2^{33.2}$	$2^{20.6}$	$2^{176.1}$	$2^{612.0}$	$2^{221.8}$	$2^{27.3}$	$2^{375.1}$	$2^{150.7}$
(approx.)	BIC	2^{71}	2^{23}	2^{10}	2^{188}	2^{330}	2^{180}	2^{17}	2^{216}	2^{111}
Worst-case		2^{210}	2^{90}	2^{42}	2^{272}	2^{1441*}	2^{506}	2^{72}	2^{727*}	2^{272}

[de Campos and Ji 2011]

Practicalities

Empirically, the sparse AD-tree data structure is the best approach for collecting sufficient statistics.

A breadth-first score calculation strategy maximizes the impact of exponential pruning.

Caching significantly reduces runtime.

Local score calculations are easily parallelizable.

Other Notes

- **MDL is numerically the same as the Bayesian Information Criterion (BIC).**

- **There are many other decomposable scoring functions that we did not cover.**
  - **Akaike's Information Criterion (AIC), which is similar to BIC**
  - **K2, which is a form of BD**
  - **MIT, which is a type of penalized KL-divergence score**

Scoring Function Wrap-up

- **There are many scoring functions available for judging the fitness of a Bayesian network given data.**
  - **Each makes different assumptions about what *optimality* means.**
  - **Others include AIC, MIT, and others.**
- **Many scoring functions are decomposable. This will be important later.**
- **Exponential score pruning can significantly reduce the number of necessary score calculations. In practice, this drastically reduces learning times.**

Outline

- Basics of Bayesian networks (30 minutes)
- Scoring functions (30 minutes)
- ➢ Dynamic programming (30 minutes)
- Admissible heuristic search (45 minutes)
- Integer linear programming (60 minutes)
- Evaluation (15 minutes)

Exploiting Scoring Function Decomposability

Observation: All BNs are DAGs, and all DAGs have a topological order.

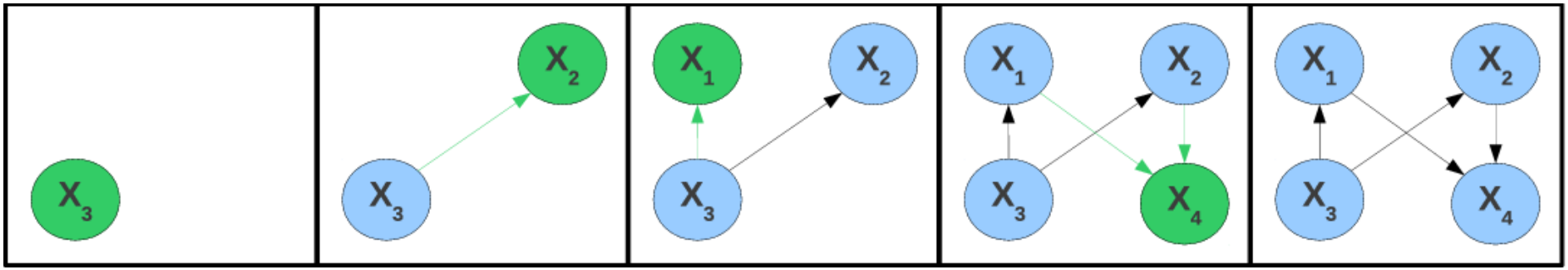
Recall decomposabile scoring functions.

$$Score(G) = \sum_{i}^{n} Score(X_i | PA_i)$$

The score of each variable depends on its parents, *but not the relationships among the parents*.

Given an ordering, we know the candidate parents for each variable.

Dynamic Programming Decomposition



Begin with a single variable.

Pick one variable as leaf. Find its optimal parents.

Pick another leaf. Find its optimal parents from current.

Continue picking leaves and finding optimal parents.

$$Score(\mathbf{U}) = \min_{X \in \mathbf{V}} Score(\mathbf{U} \setminus \{X\}) + BestScore(X, \mathbf{U})$$

$$BestScore(X, \boldsymbol{U}) = \min_{PA_X \subseteq \mathbf{U} \setminus \{X\}} Score(X|PA_X)$$

[Silander and Myllymaki 2006]

Dynamic Programming for BNSL

Goal: Given a scoring function Score, we want to find

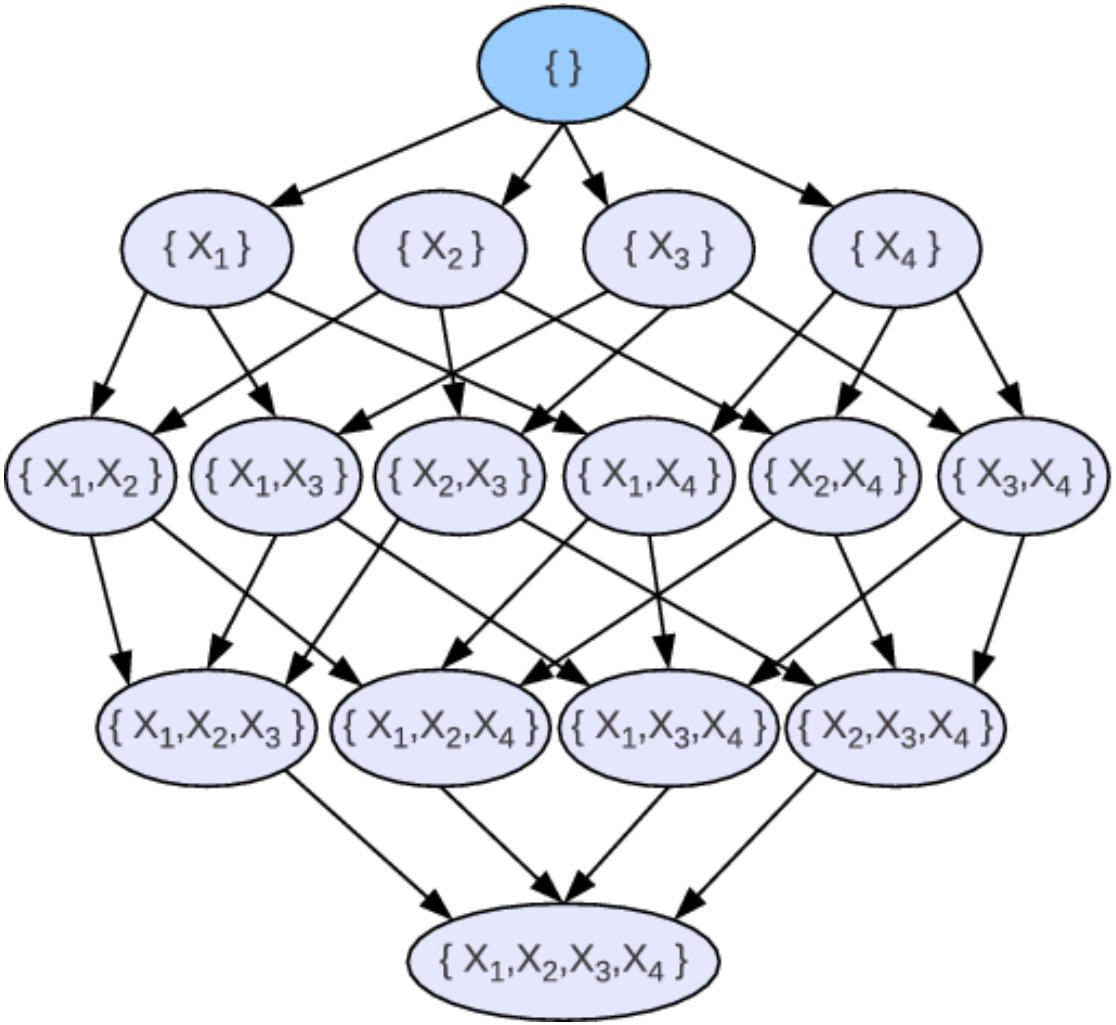
$$G^* = \underset{G}{\operatorname{argmin}}\, Score(G : \boldsymbol{D})$$

Observation: All BNs are DAGs, and all DAGs have a topological ordering. Therefore, G* has an ordering.

New Goal: Find the ordering of G*.

Problem: There are n! orderings.

Learning Optimal Bayesian Networks via DP



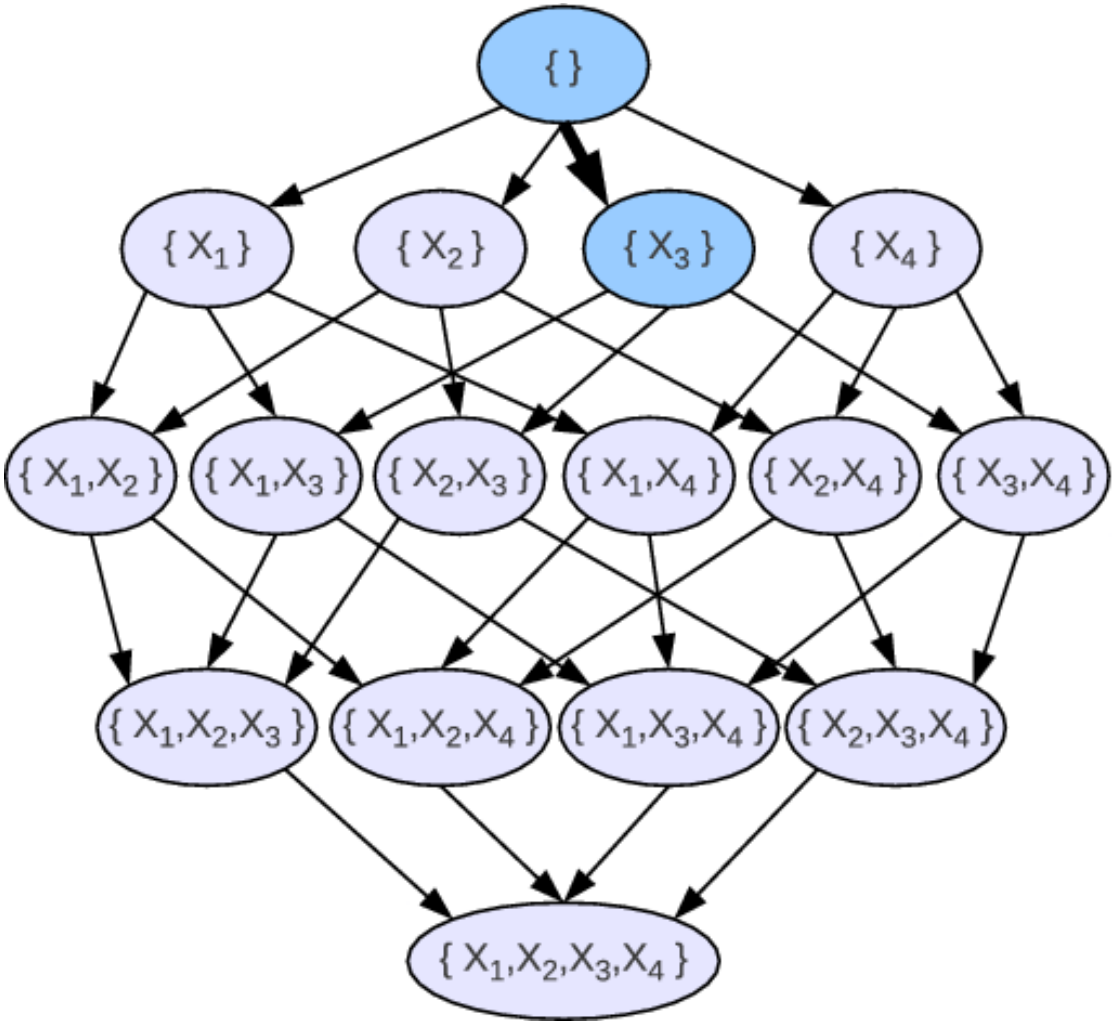
A path from the start to the goal induces an ordering on the variables.

Based on the ordering, we can calculate the optimal network with the recurrences.

We call this the order graph.

[Ott et al. 2004, Yuan et al. 2011]

Learning Optimal Bayesian Networks via DP



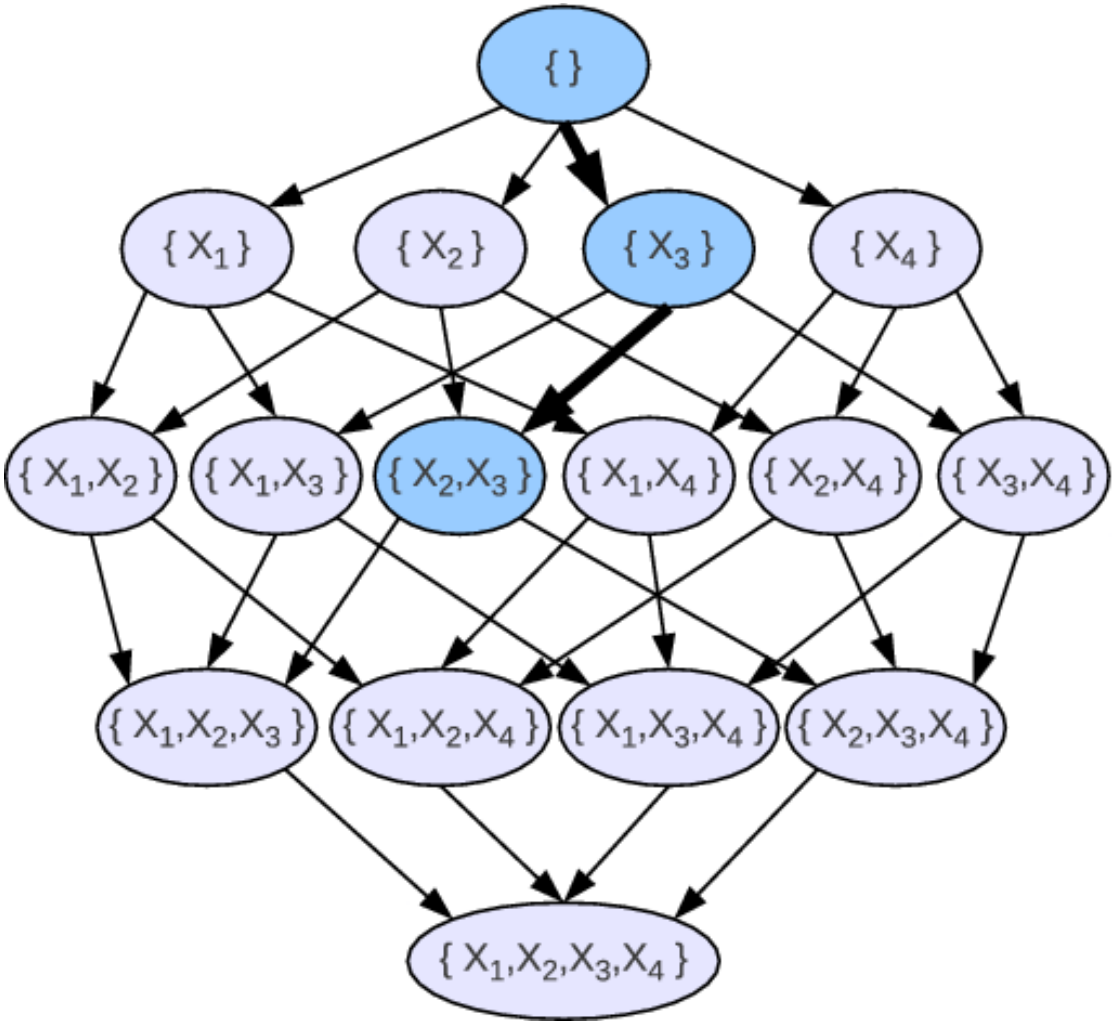
A path from the start to the goal induces an ordering on the variables.

Based on the ordering, we can calculate the optimal network with the recurrences.

We call this the order graph.

[Ott et al. 2004, Yuan et al. 2011]

Learning Optimal Bayesian Networks via DP



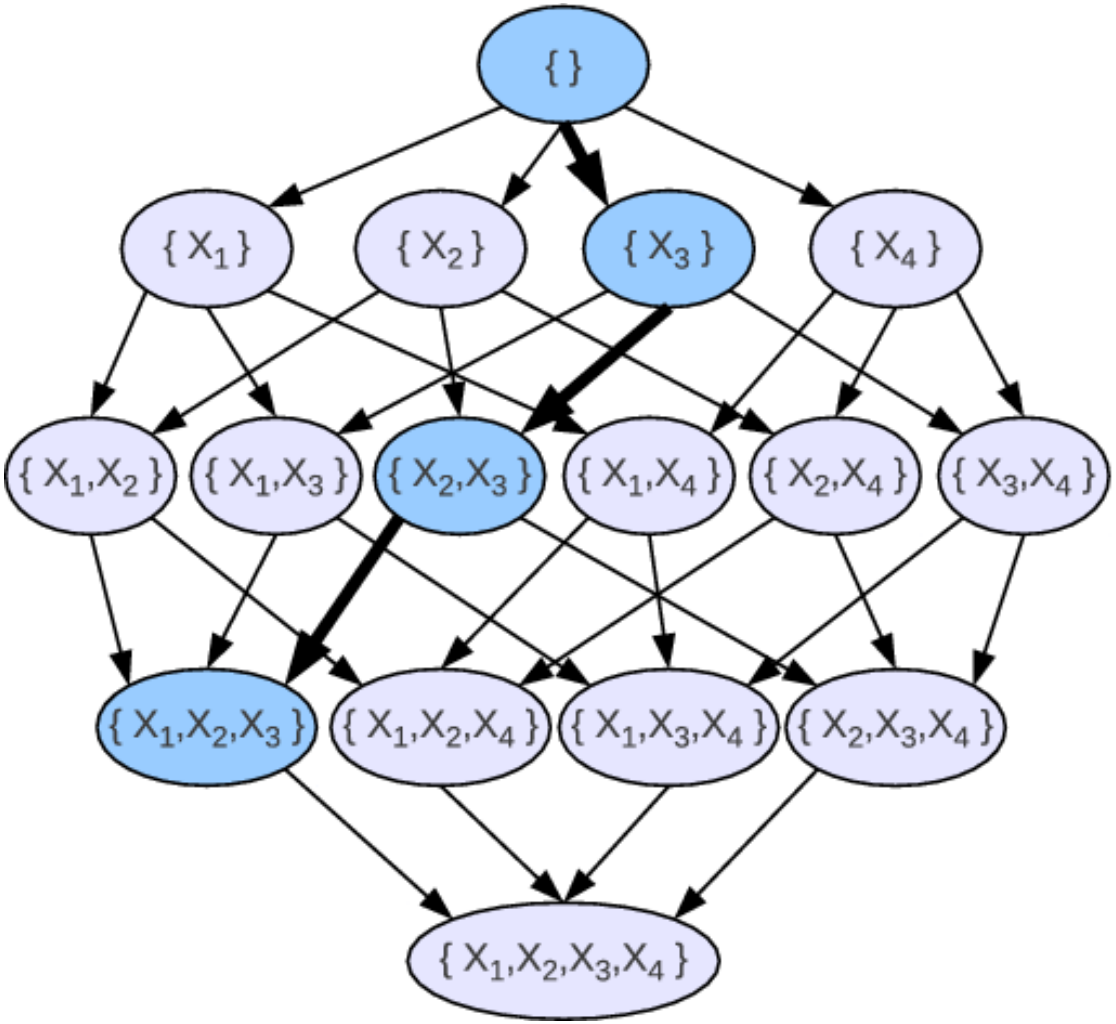
A path from the start to the goal induces an ordering on the variables.

Based on the ordering, we can calculate the optimal network with the recurrences.

We call this the order graph.

[Ott et al. 2004, Yuan et al. 2011]

Learning Optimal Bayesian Networks via DP



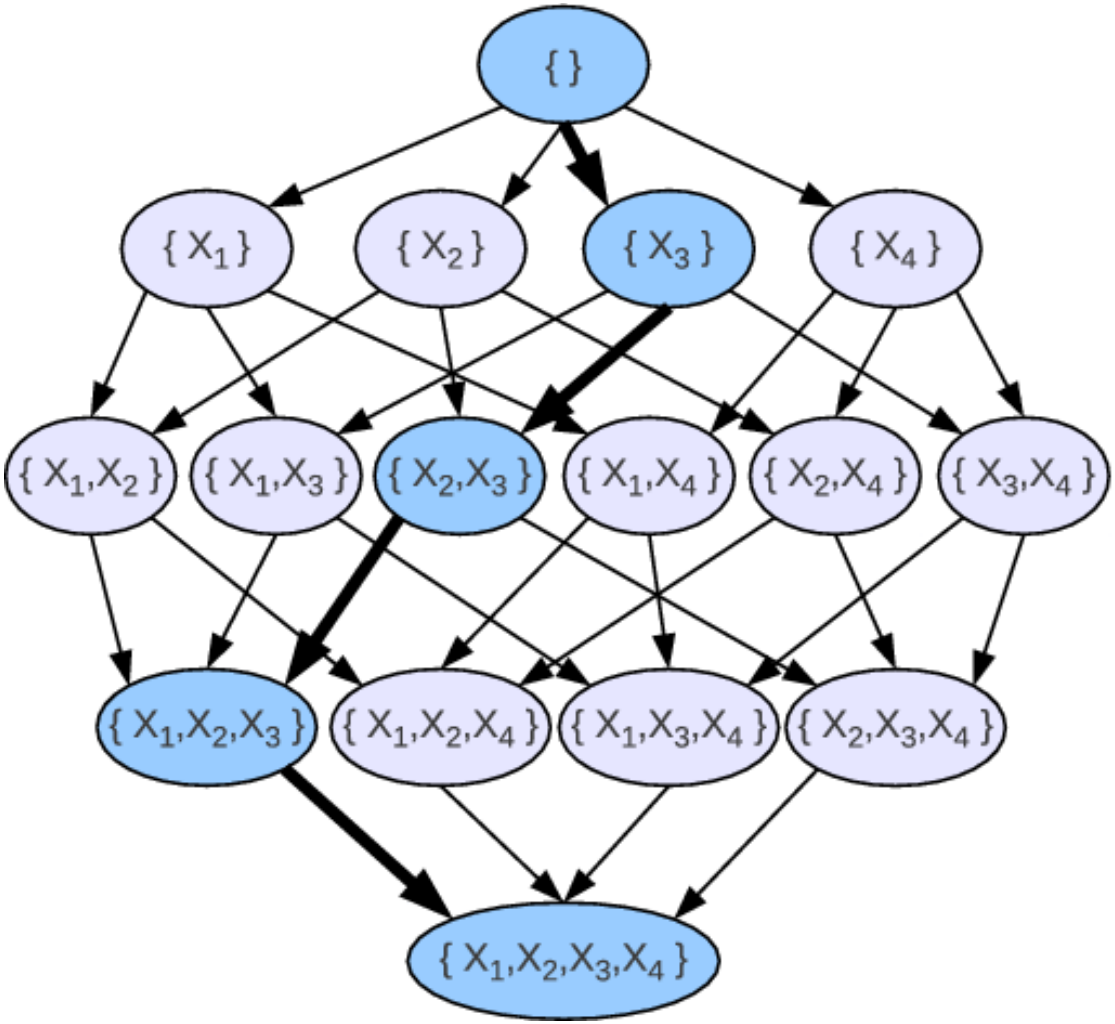
A path from the start to the goal induces an ordering on the variables.

Based on the ordering, we can calculate the optimal network with the recurrences.

We call this the order graph.

[Ott et al. 2004, Yuan et al. 2011]

Learning Optimal Bayesian Networks via DP



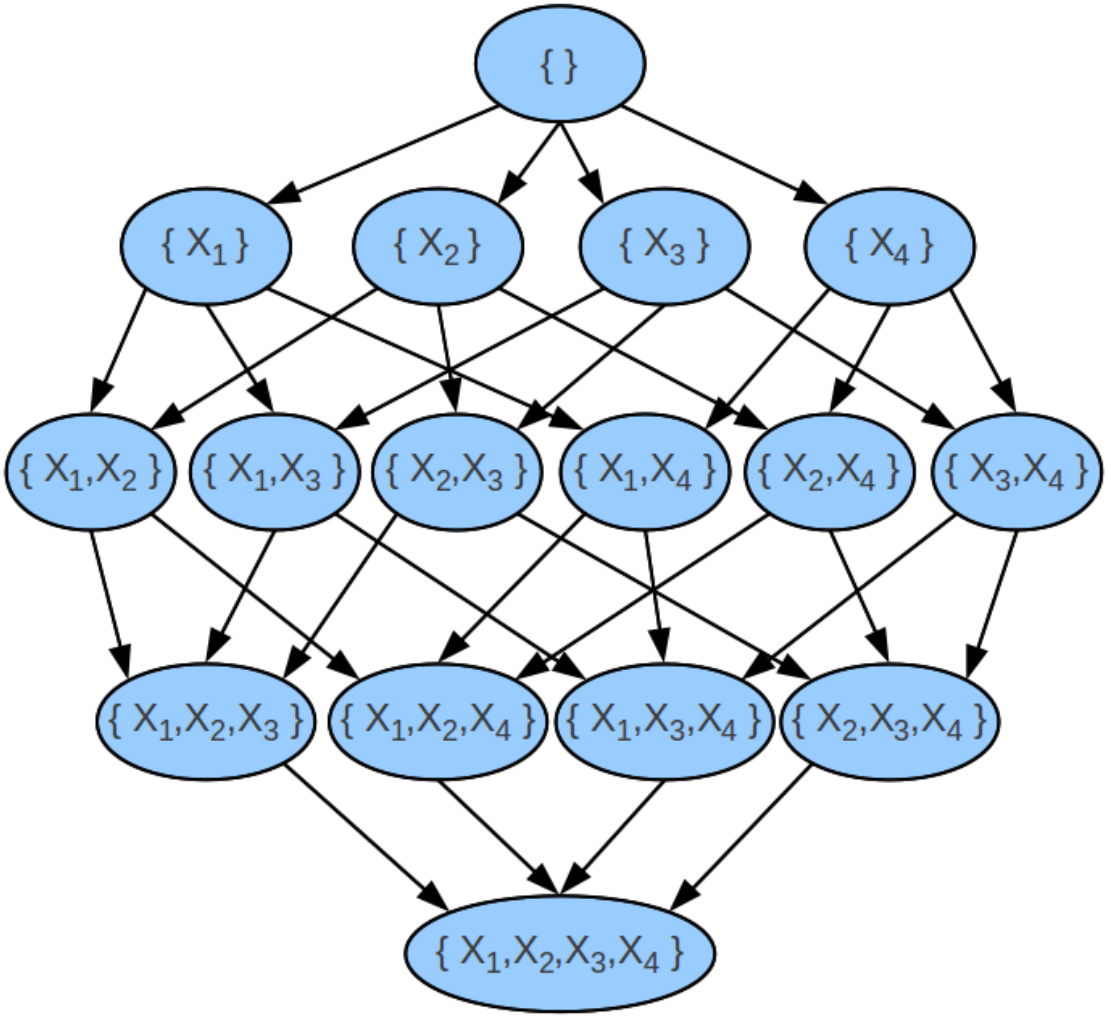
A path from the start to the goal induces an ordering on the variables.

Based on the ordering, we can calculate the optimal network with the recurrences.

We call this the order graph.

[Ott et al. 2004, Yuan et al. 2011]

Learning Optimal Bayesian Networks via DP



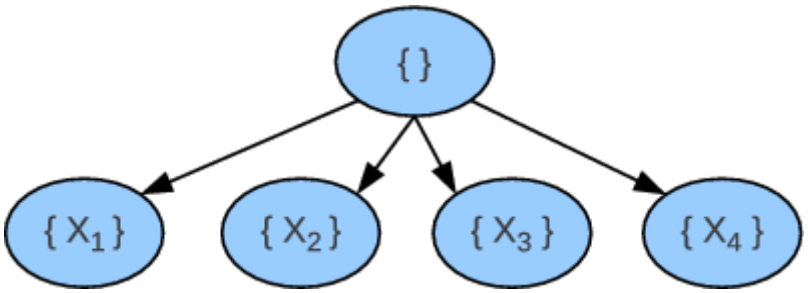
Problem: There are still $n!$ orderings.

Observation: Scoring decomposability means parent selections are independent.

Solution: Decouple paths into and out of the nodes.

[Ott et al. 2004, Yuan et al. 2011]

Memory-efficient Dynamic Programming



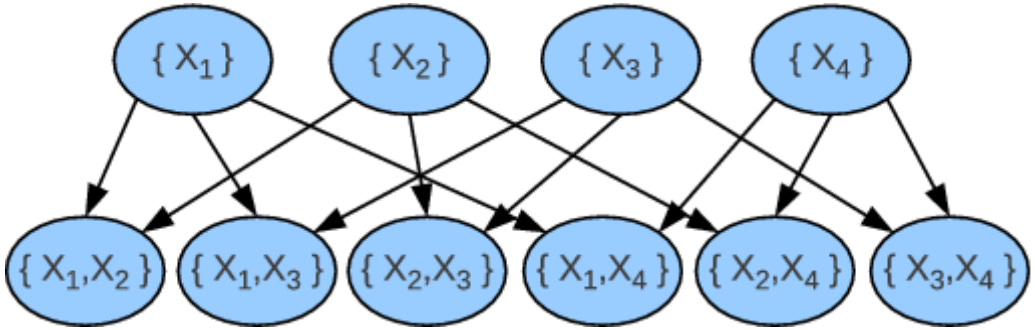
Problem: There are still 2^n nodes.

Observation: Only nodes in the previous layer are required.

Solution: Only store a layer as long as necessary.

[Malone et al. 2011]

Memory-efficient Dynamic Programming



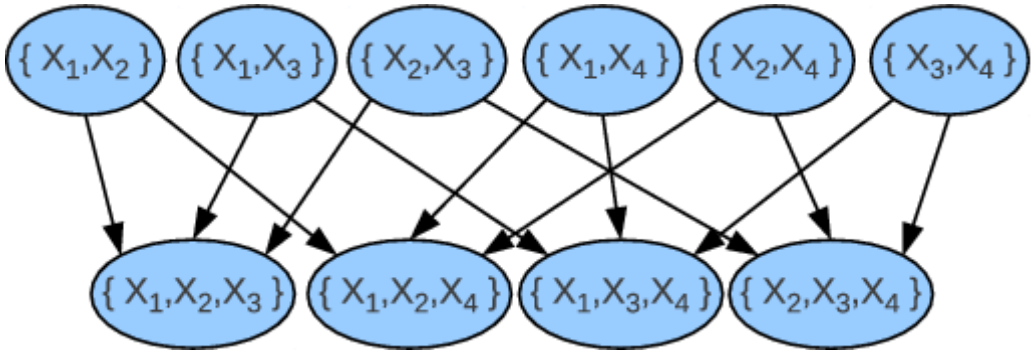
Problem: There are still 2^n nodes.

Observation: Only nodes in the previous layer are required.

Solution: Only store a layer as long as necessary.

[Malone et al. 2011]

Memory-efficient Dynamic Programming



Problem: There are still 2^n nodes.

Observation: Only nodes in the previous layer are required.

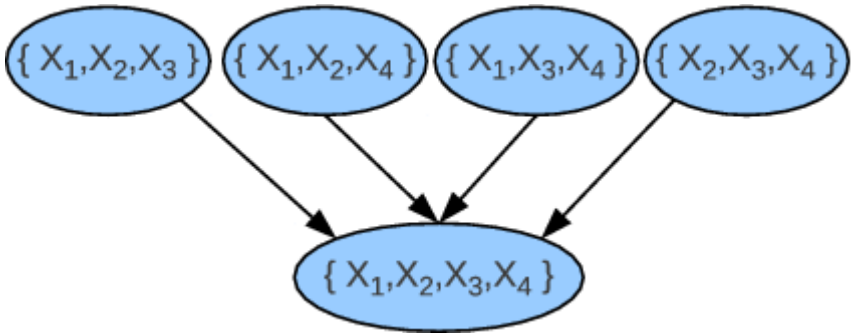
Solution: Only store a layer as long as necessary.

[Malone et al. 2011]

Memory-efficient Dynamic Programming

Problem: There are still 2^n nodes.

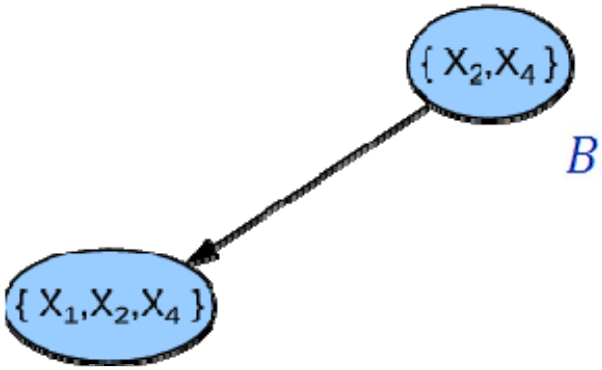
Observation: Only nodes in the previous layer are required.



Solution: Only store a layer as long as necessary.

[Malone et al. 2011]

How can we calculate the edge costs?



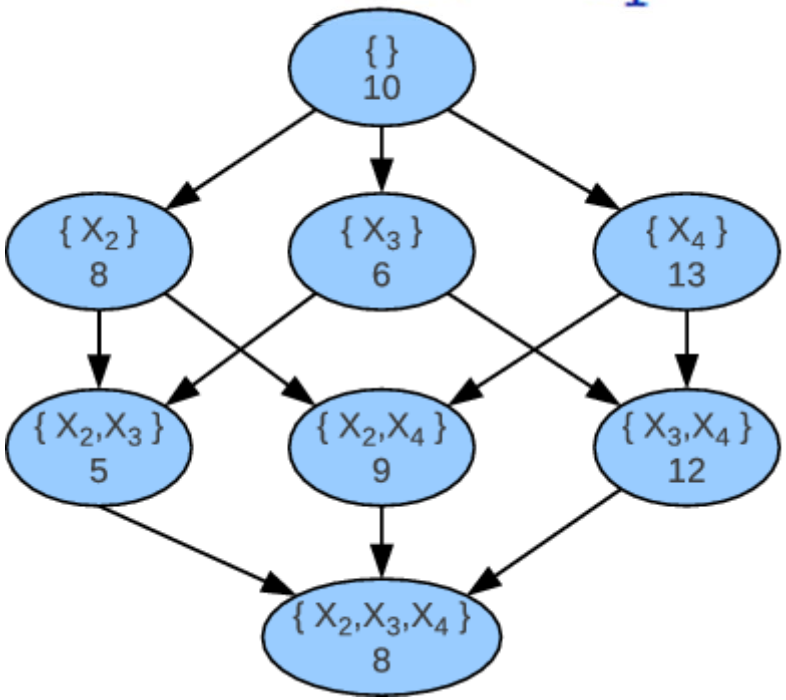
$$BestScore(X, \boldsymbol{U}) = \min_{PA_X \subseteq \mathbf{U} \setminus \{X\}} Score(X|PA_X)$$

$$BestScore(X_1, \{X_2, X_4\}) = \min_{PA_{X_1} \subseteq \{X_2, X_4\}} Score(X_1|PA_{X_1})$$

$$Score(X_1|PA_{X_1})$$

"Solution": Search through all of the local scores and find the best

Problem: Quite expensive because we must perform this operation for all edges.

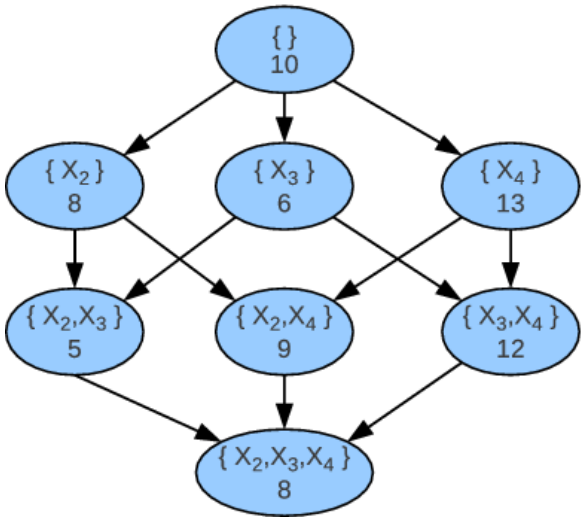


How can we *efficiently* calculate the edge costs?

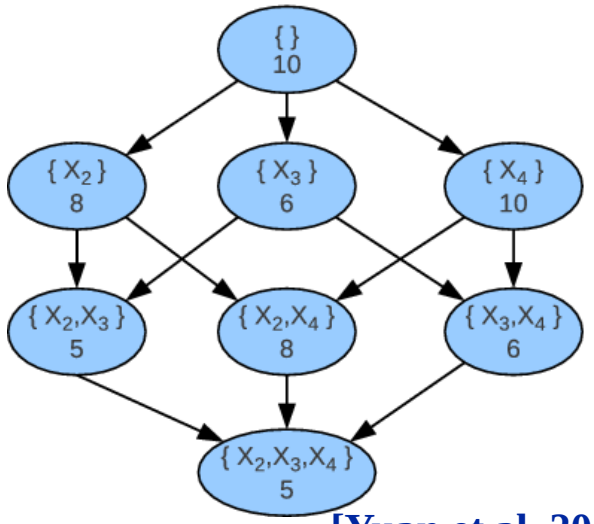
Theorem: Say $PA_i \subset PA'_i$ and $Score(X_i|PA_i) > Score(X|PA'_i)$. Then PA'_i is not optimal for X_i.

Solution: Before main search, propagate optimal scores through the parent graph and store as hash table.

$Score(X_1|PA_1)$



$BestScore(X_1|PA_1)$



[Yuan et al. 2011]

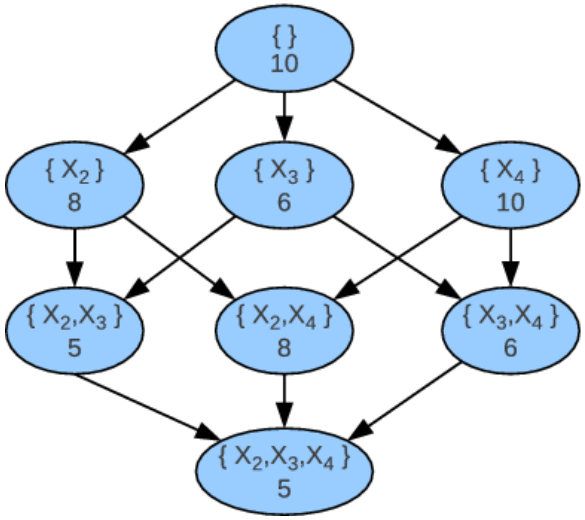
Pruning Parent Graphs

Observation: Most parent sets are not optimal.

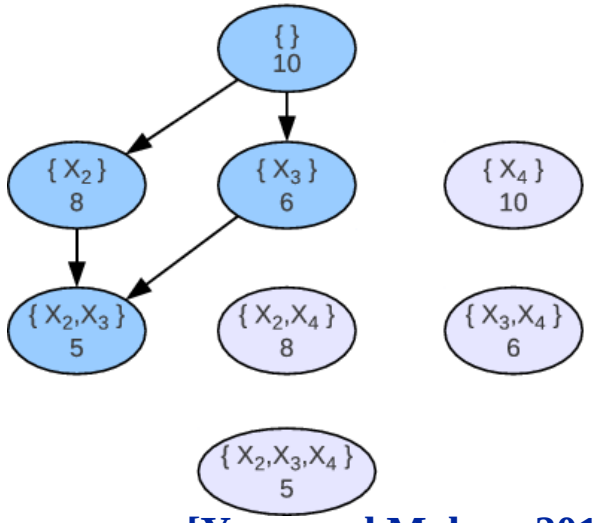
Problem: We create parent graph nodes anyway. So parent graphs are always size 2^{n-1}.

Solution: Only create nodes for parent sets which could possibly be optimal.

$BestScore(X_1|PA_1)$



$PossiblyOptimal(X_1|PA_1)$



[Yuan and Malone 2012]

Efficient Algorithms for Sparse Parent Graphs

Problem: How can we efficiently lookup BestScore?

Solution: Use a bitwise representation.

$PossiblyOptimal(X_1|PA_1)$

X_2,X_3	X_3	X_2	ϕ
5	6	8	10

Create bitsets of which scores use which parents.

$allparents_{X_1}$	1	1	1	1
$parents_{X_1}^{X_2}$	1		1	
$parents_{X_1}^{X_3}$	1	1		
$parents_{X_1}^{X_4}$				

Use these to calculate BestScore, e.g. **[Yuan and Malone 2012]**

$$BestScore(X_1|X_2,X_4) = first(allparents_{X_1} \& \sim parents_{X_1}^{X_3})$$

Constraints and Sparse Parent Graphs

Observation: In many domains, we know something about some of the variables.

We can easily incorporate local constraints by simply omitting parent sets which violate the constraints.

If we perform a BestScore calculation and find no set bits, then that path is not consistent with the constraints.

Dynamic Programming Wrap-up

- **Scoring function decomposability allows independent parent set selections, given an ordering.**
- **After finding an optimal subnetwork, we no longer need information about how we found it.**
- **These observations allow us to formulate the search space in terms of a dynamic programming lattice.**
- **Efficient data structures are necessary for making individual parent set selections.**

Outline

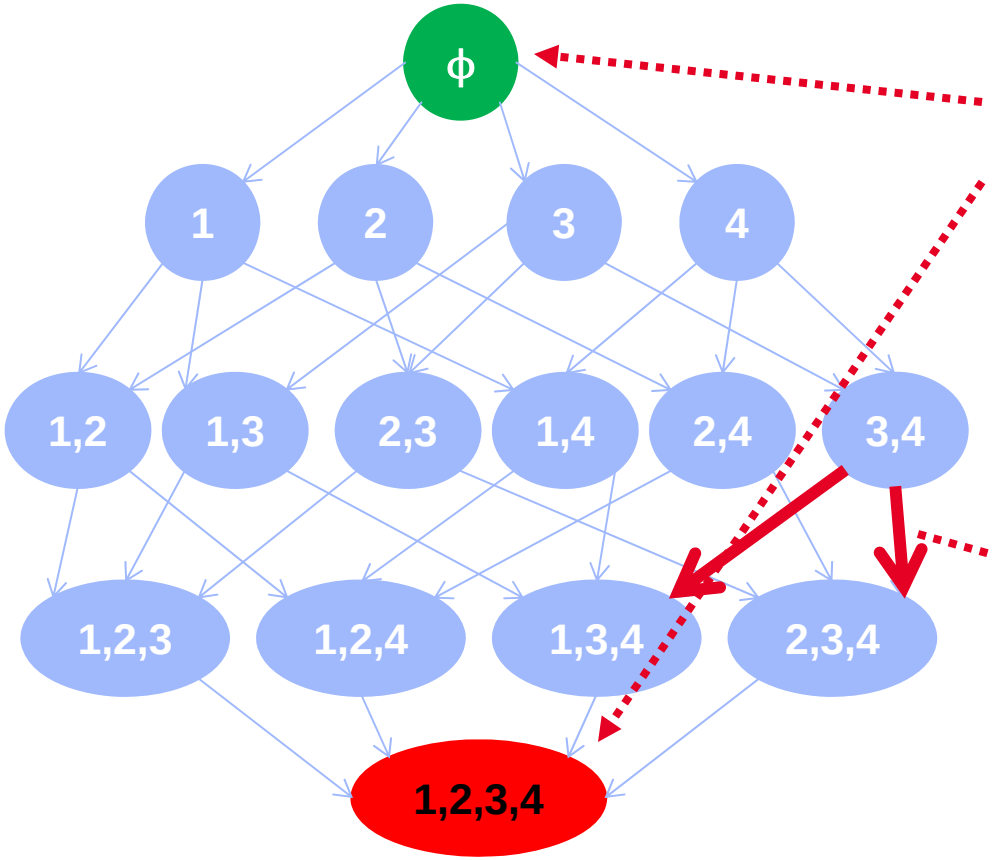
- Basics of Bayesian networks (30 minutes)
- Scoring functions (30 minutes)
- Dynamic programming (30 minutes)
- ➢ Admissible heuristic search (45 minutes)
- Integer linear programming (60 minutes)
- Empirical evaluations (15 minutes)

Graph search formulation

- **Share the same principle with dynamic programming**
- **Formulate the learning task as a shortest path problem**
  - **The shortest path solution to a graph search problem corresponds to an optimal Bayesian network**

[Yuan, Malone, Wu, IJCAI-11]

Search graph (Order graph)



Formulation:

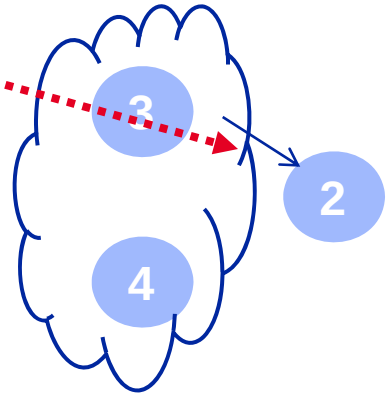
Search space: Variable subsets

Start node: Empty set

Goal node: Complete set

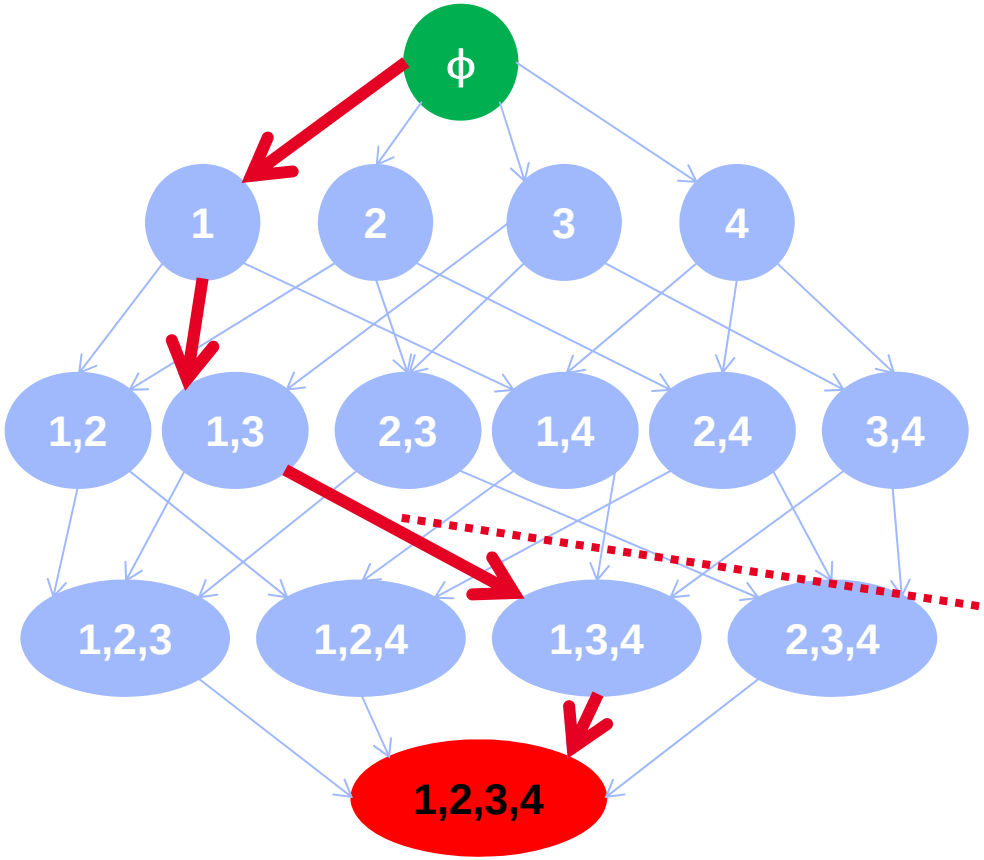
Edges: Select parents

Edge cost: $\mathrm{BestScore}(X,U)$ for edge $U \rightarrow U \cup \{X\}$



[Yuan, Malone, Wu, IJCAI-11]

Search graph (Order graph)



Formulation:

Search space: Variable subsets

Start node: Empty set

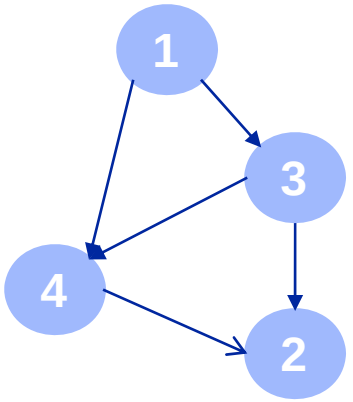
Goal node: Complete set

Edges: Select parents

Edge cost: $\mathrm{BestScore}(X,U)$ for edge $U \rightarrow U \cup \{X\}$

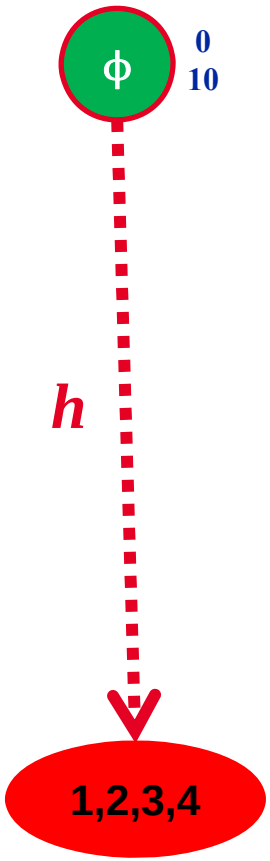
Task: find the shortest path between start and goal nodes

1,3,4,2



[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

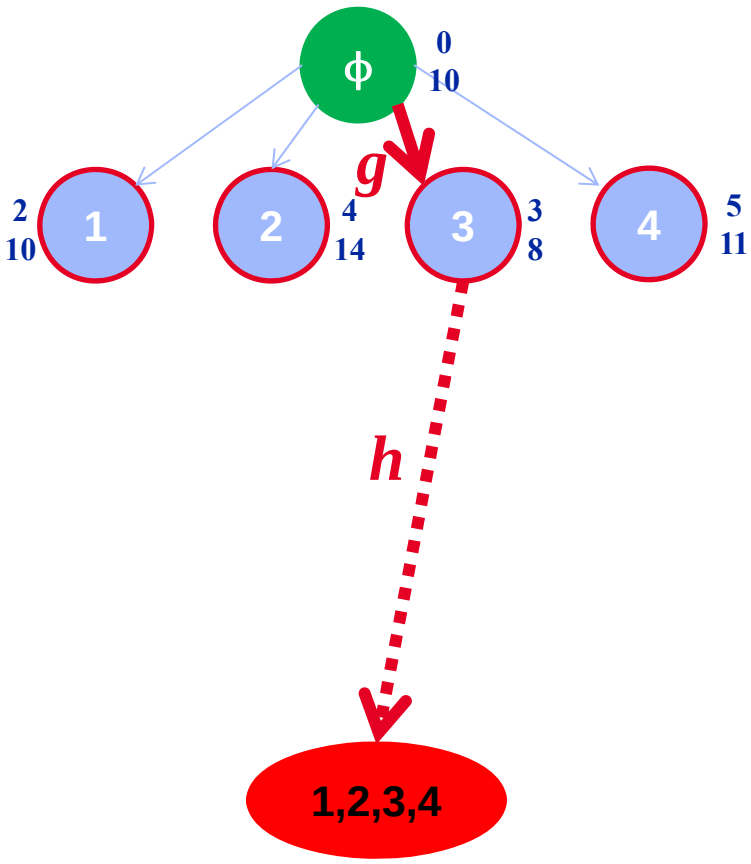
$h(U) = estimated\ distance\ to\ goal$

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

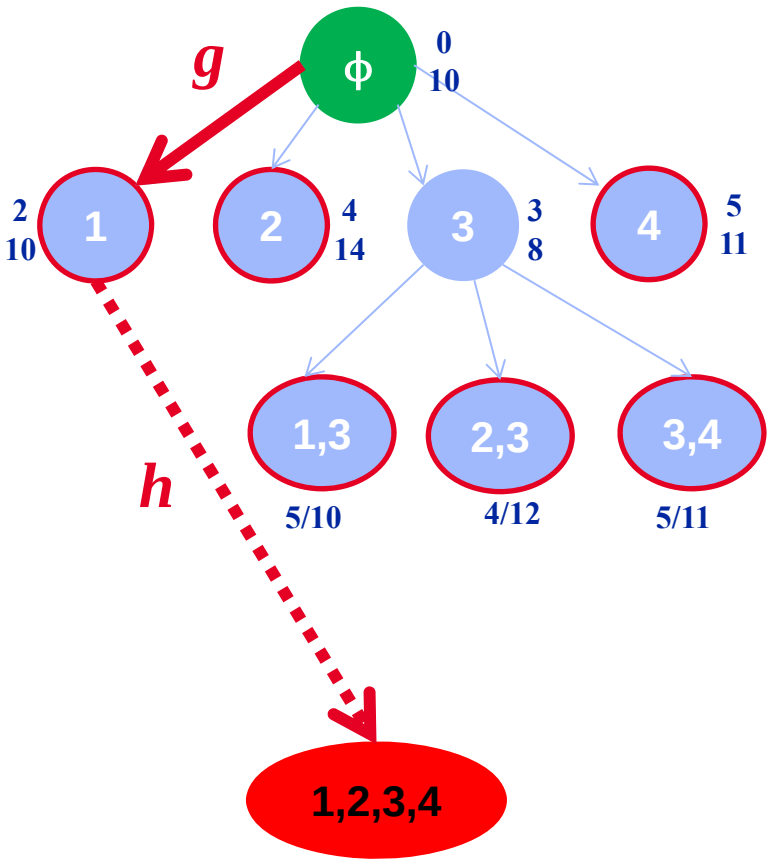
$h(U) = estimated\ distance\ to\ goal$

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

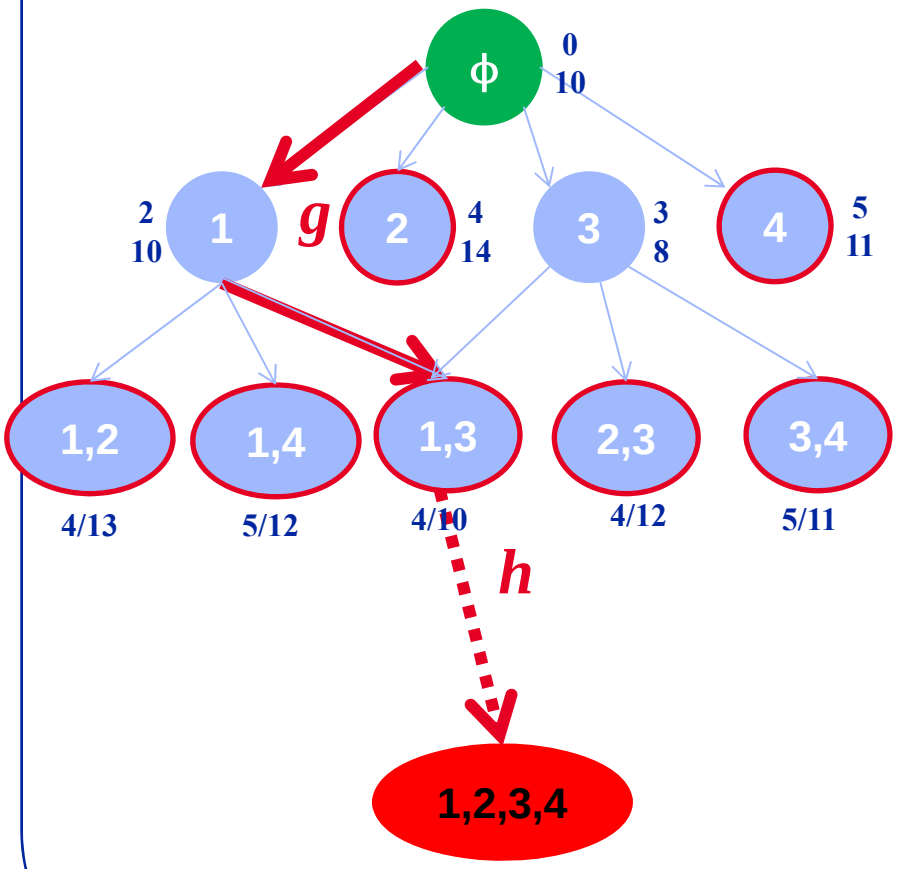
$h(U) =$ *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

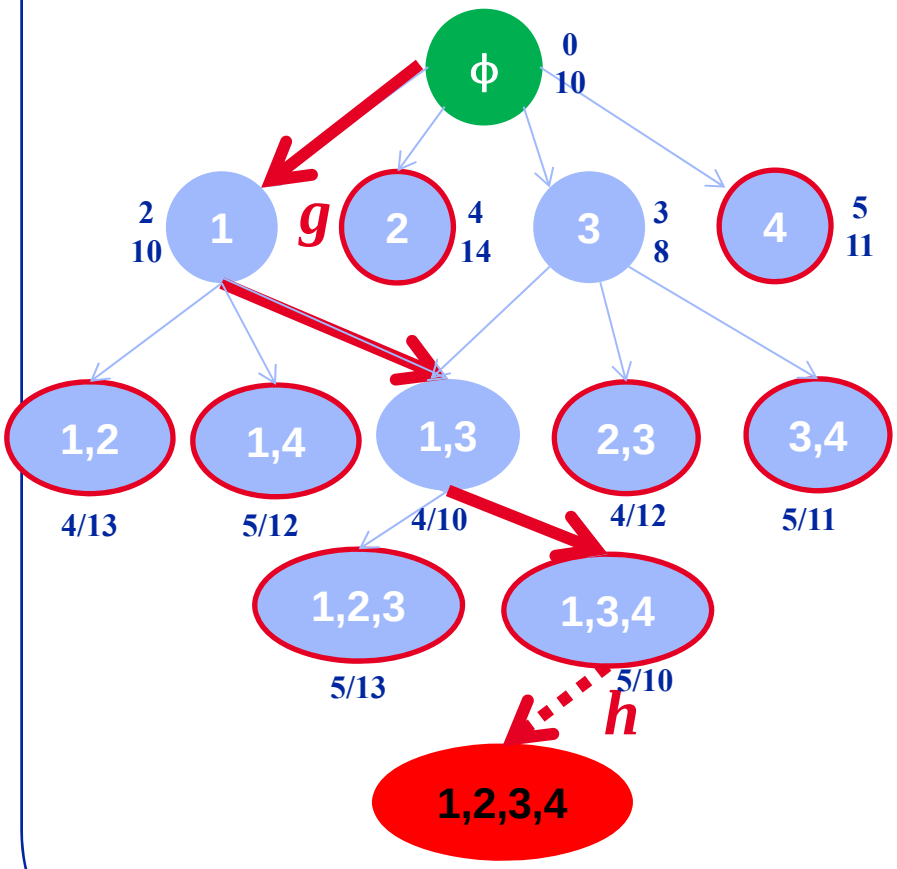
$h(U) =$ *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

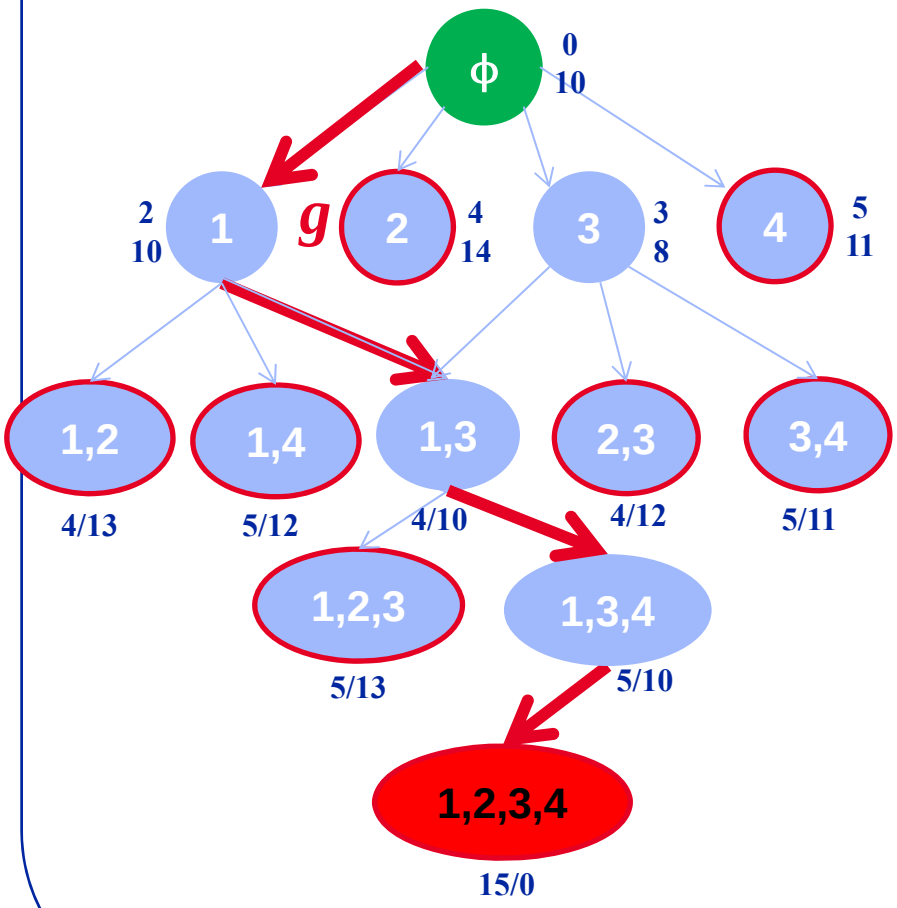
$h(U) =$ *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

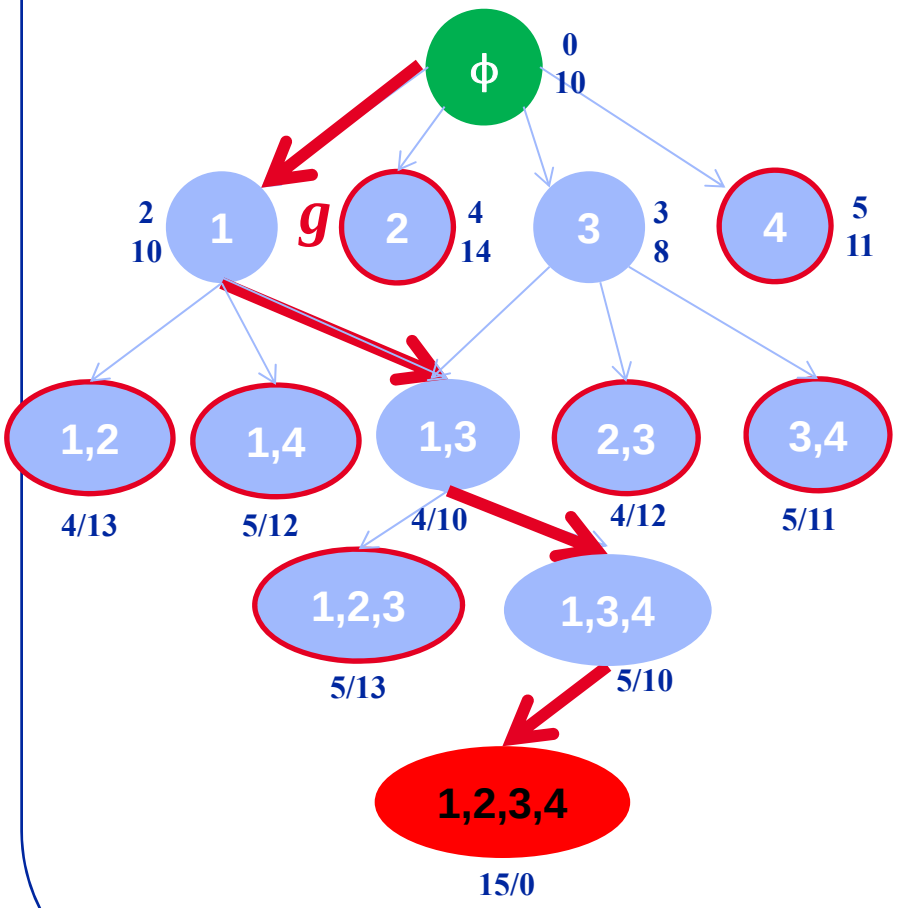
$h(U) =$ *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

A* algorithm



A* search: Expands the nodes in the order of quality: $f=g+h$

$g(U) = Score(U)$

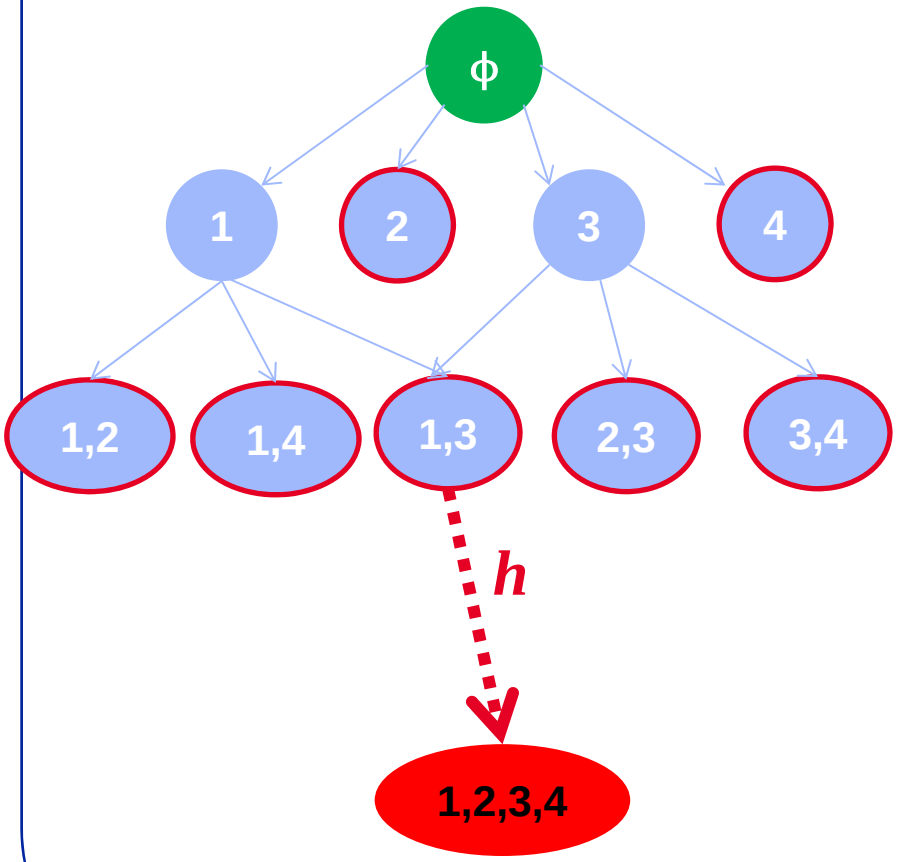
$h(U) =$ *estimated distance to goal*

Notation:

g:	g-cost
h:	h-cost
Red shape-outlined:	open nodes
No outline:	closed nodes

[Yuan, Malone, Wu, IJCAI-11]

Simple heuristic

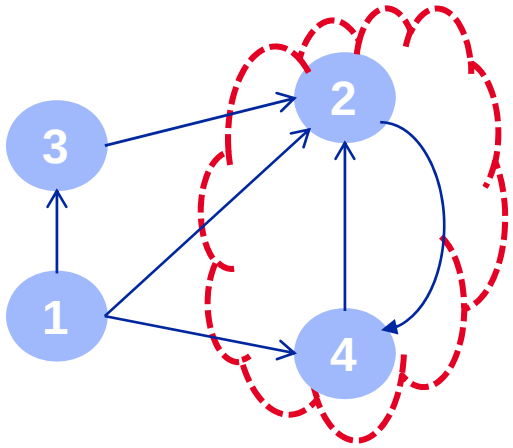


A* search: Expands the nodes in the order of quality: $f=g+h$

$$g(U) = Score(U)$$

$$h(U) = \sum_{X \in V \setminus U} BestScore(X, V \setminus \{X\})$$

***h*({1,3}):**



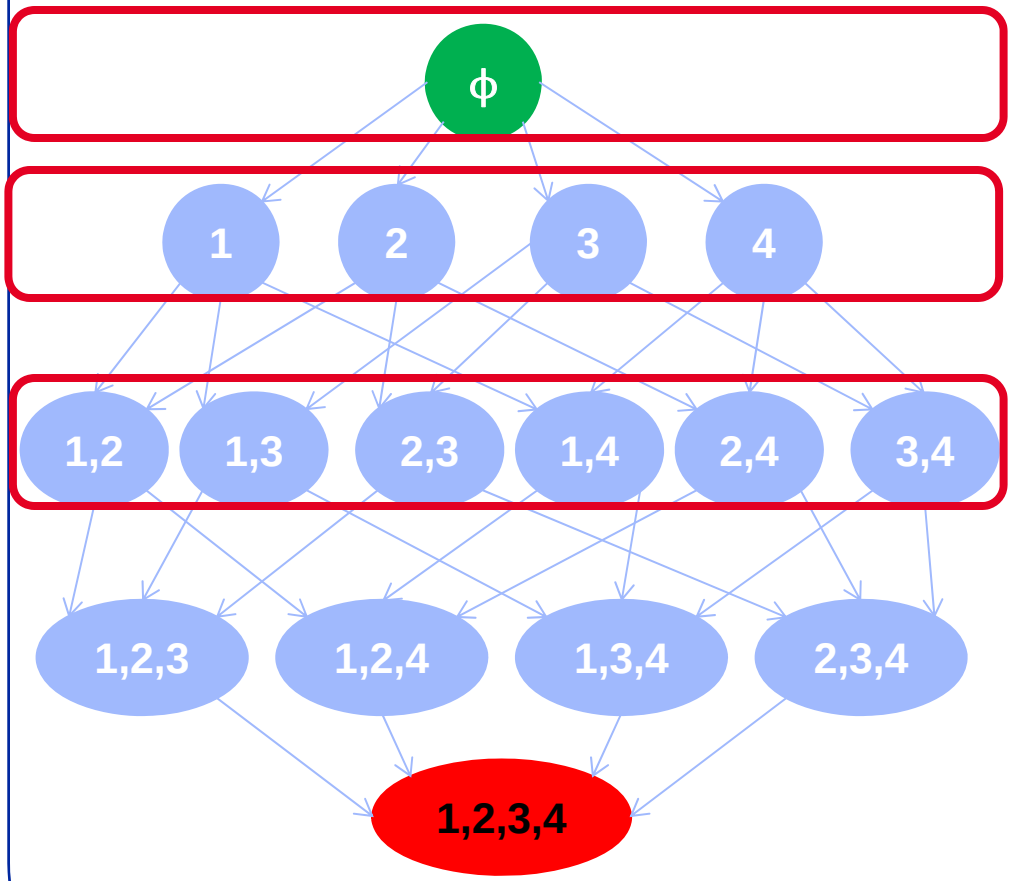
[Yuan, Malone, Wu, IJCAI-11]

Properties of the simple heuristic

- **Theorem: The simple heuristic function *h* is admissible**
  - **Optimistic estimation: never overestimate the true distance**
  - **Guarantees the optimality of A***
- **Theorem: *h* is also consistent**
  - **Satisfies triangular inequality**
  - **Consistency => admissibility**
  - **Guarantees the optimality of *g* cost of any node to be expanded**

[Yuan, Malone, Wu, IJCAI-11]

BFBnB algorithm

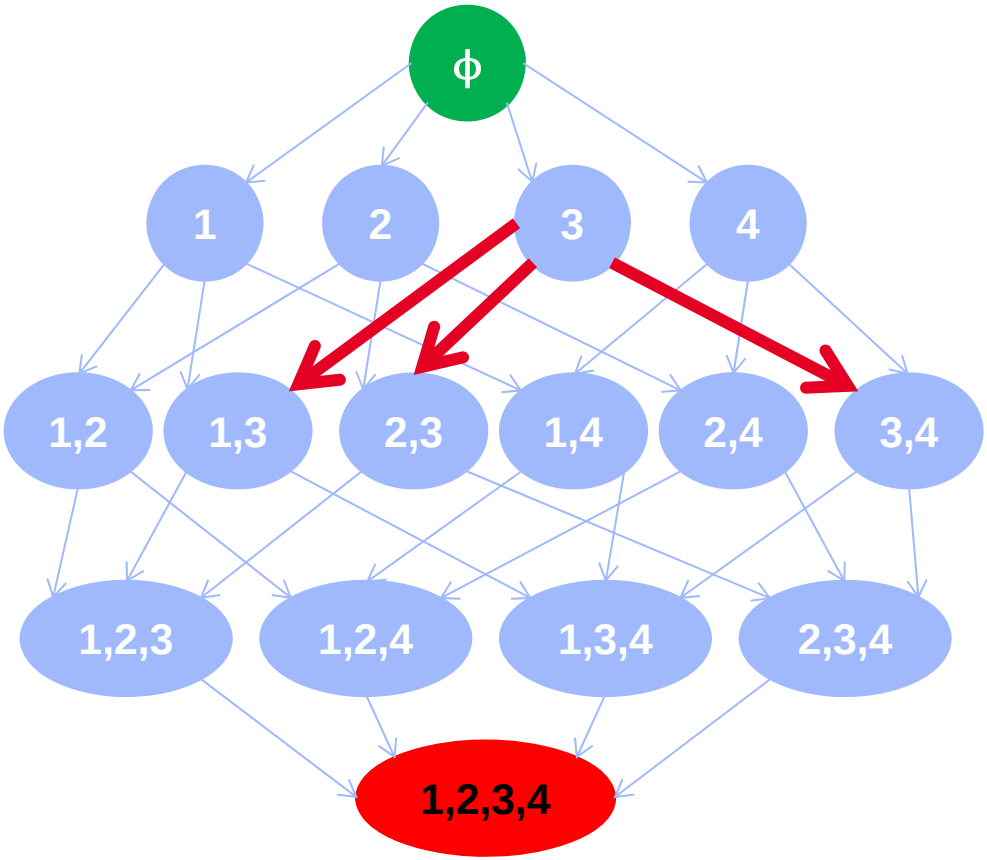


Breadth-first branch and bound search (BFBnB):

- **Motivation:**
  - Exponential-size order&parent graphs
- **Observation:**
  - Natural layered structure
- **Solution:**
  - Search one layer at a time

[Malone, Yuan, Hansen, UAI-11]

BFBnB algorithm

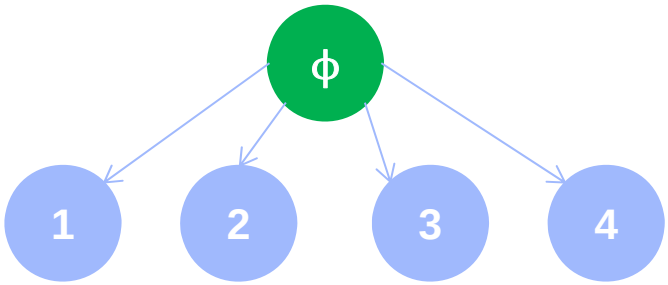


Breadth-first branch and bound search (BFBnB):

- **Motivation:**
  - Exponential-size order&parent graphs
- **Observation:**
  - Natural layered structure
- **Solution:**
  - Search one layer at a time

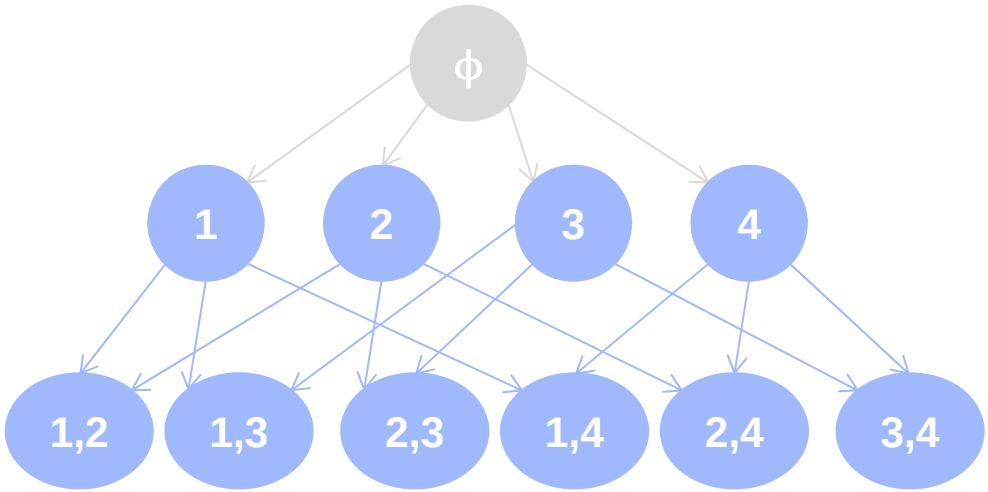
[Malone, Yuan, Hansen, UAI-11]

BFBnB algorithm



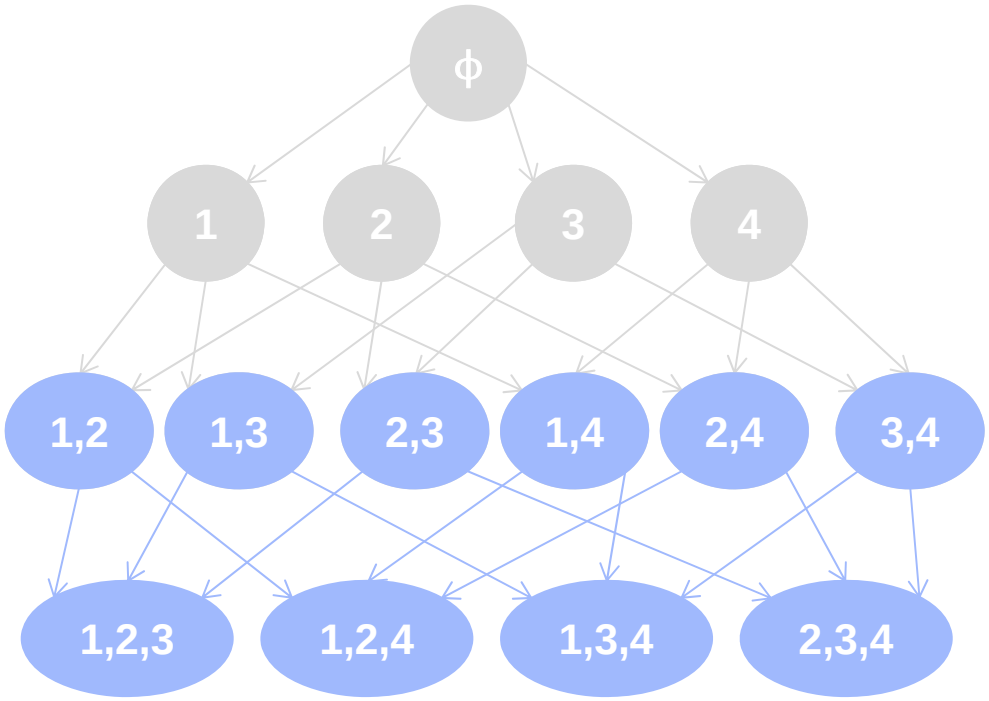
[Malone, Yuan, Hansen, UAI-11]

BFBnB algorithm



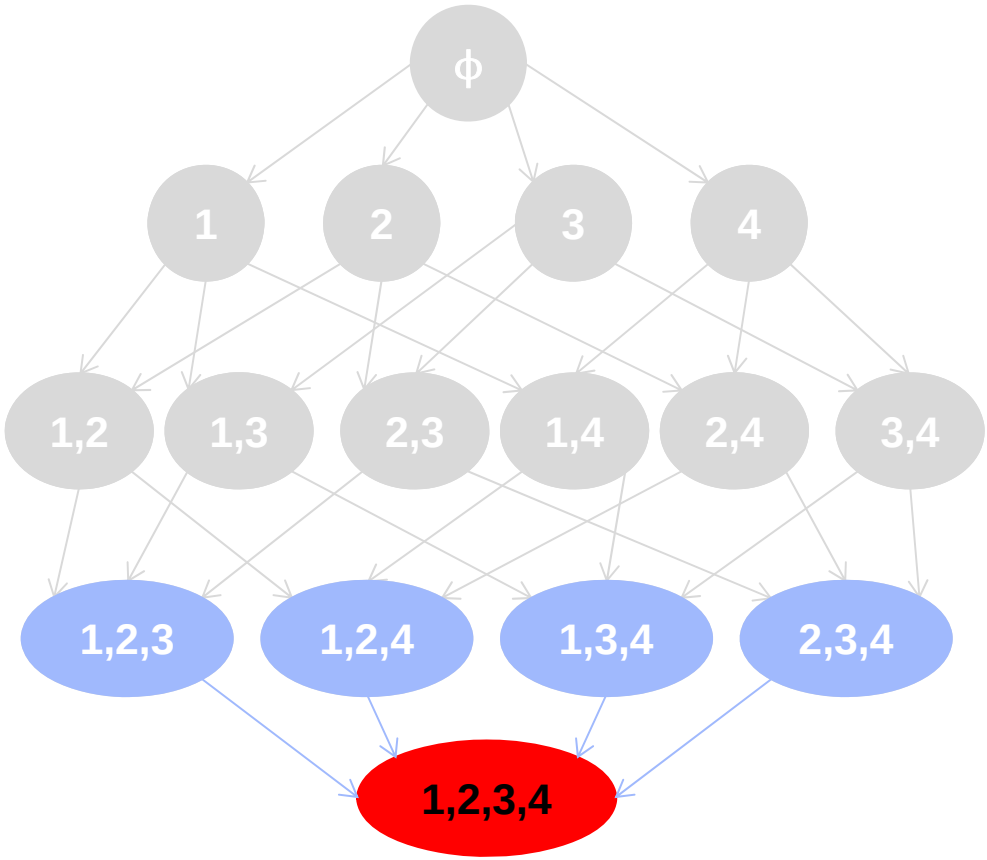
[Malone, Yuan, Hansen, UAI-11]

BFBnB algorithm



[Malone, Yuan, Hansen, UAI-11]

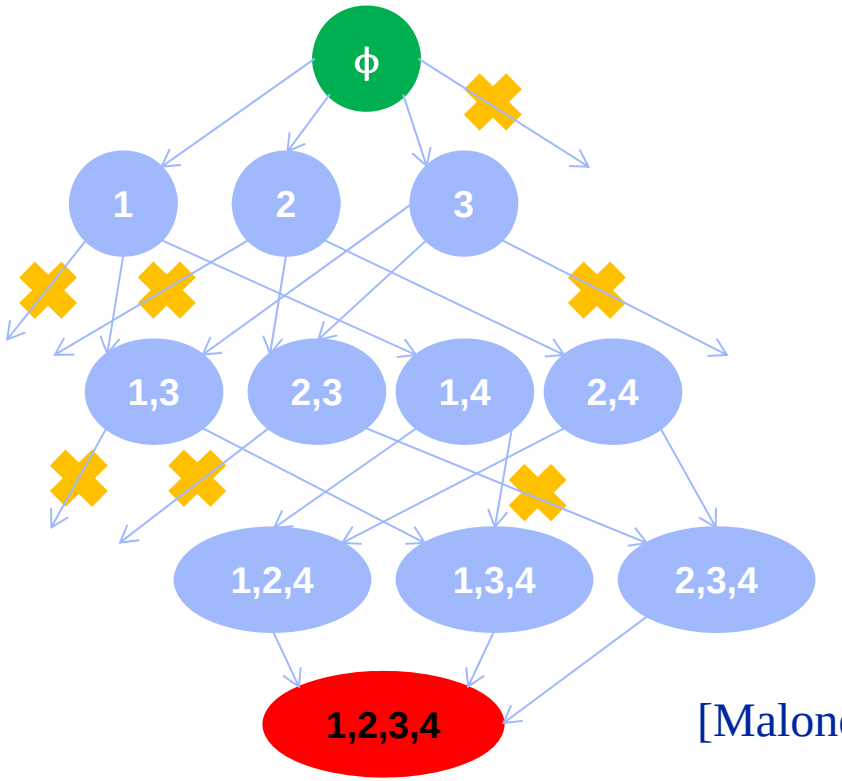
BFBnB algorithm



[Malone, Yuan, Hansen, UAI-11]

Pruning in BFBnB

- For pruning, estimate an upper bound solution before search
  - Can be done using greedy local search
- Prune a node when *f*-cost > upper bound

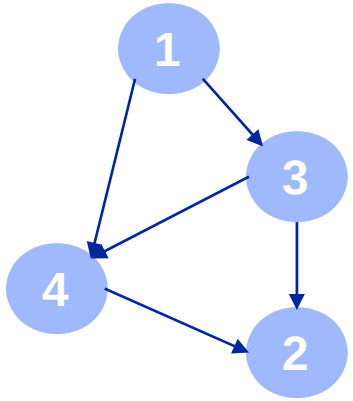


[Malone, Yuan, Hansen, UAI-11]

Critiquing the simple heuristic

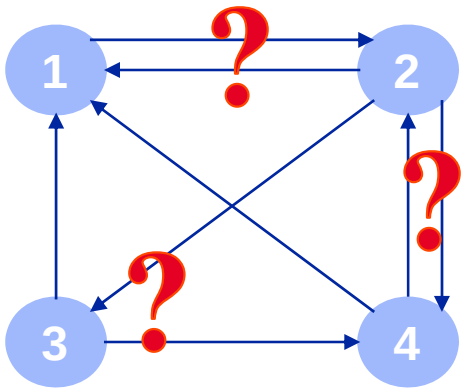
- **Drawback of the simple heuristic**
  - Let each variable to choose optimal parents from all the other variables
  - Completely relaxes the acyclicity constraint

Bayesian network



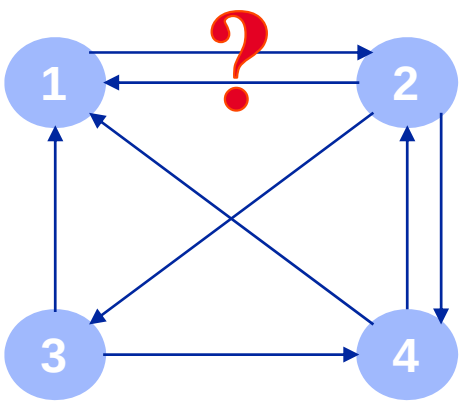
Relaxation

Heuristic estimation



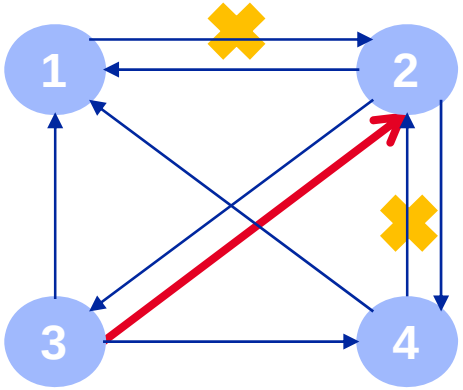
Potential solution

- **Breaking the cycles to obtain tighter heuristic**



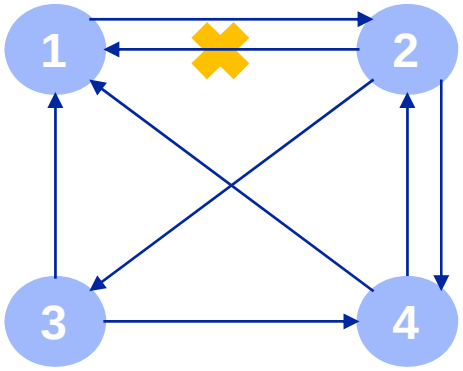
BestScore(*1*, {*2*,*3*,*4*})={*2*,*3*,*4*}
+
BestScore(*2*, {*1*,*3*,*4*})={*1*,*4*}

Delete 1→2



BestScore(*1*, {*2*,*3*,*4*})
+
BestScore(*2*, {*3*,*4*})={*3*}

Delete 2→1



BestScore(*1*, {*3*,*4*})={*3*,*4*}
+
BestScore(*2*, {*1*,*3*,*4*})

min ⇒ *c*({*1*,*2*})

[Yuan, Malone, UAI-12]

k-cycle conflict heuristic

- **Compute costs for all 2-variable groups**
  - **Each group, called a pattern, has a tighter score**

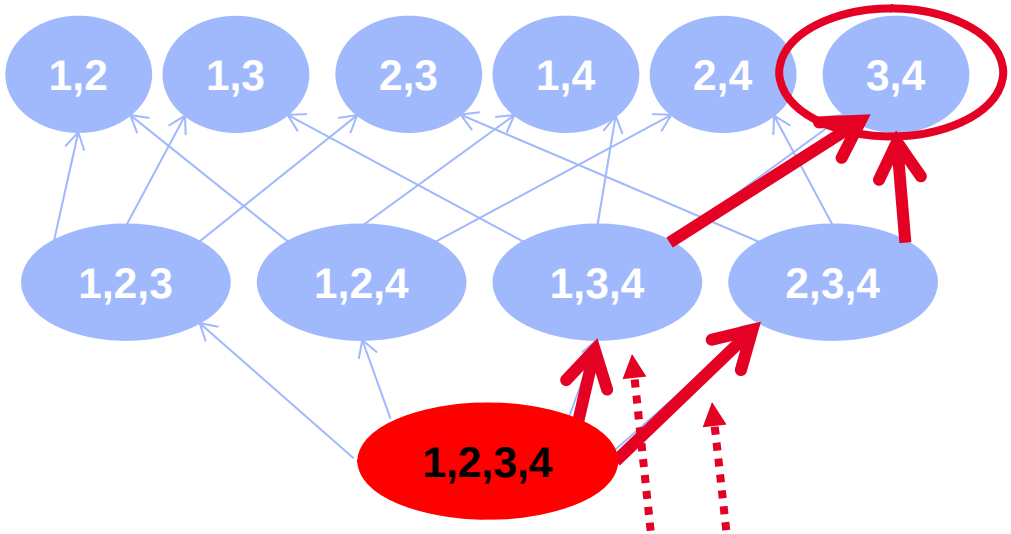
- ***k*-cycle conflict heuristic**
  - **Based on dynamically partitioned pattern database** [Felner et al. 2004]
  - **Avoid cycles for variable groups with size up to *k***

- **Two research issues**
  - **How to compute costs for all the patterns?**
  - **How to use the patterns?**

[Yuan, Malone, UAI-12]

Compute *k*-cycle conflict heuristic

- **Compute the pattern database with a backward breadth-first search in the order graph for *k* layers**



Reverse *g* cost:

$g^r(\{3,4\}) = \min(\boldsymbol{P1}, \boldsymbol{P2})$

P1: BestScore(*1*, {*2*,*3*,*4*})+BestScore(*2*,{*3*,*4*})

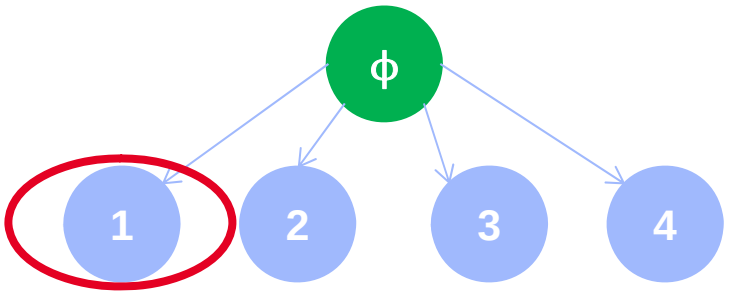
P2: BestScore(*2*, {*1*,*3*,*4*})+BestScore(*1*,{*3*,*4*})

$g^r(\{3,4\}) == c(\{1,2\})$!

[Yuan, Malone, UAI-12]

Computing heuristic value using dynamic PD

- **To compute the heuristic for a node, find a set of exclusive patterns for remaining variables and sum their costs**



$$h(\{1\}) = c(\{2,3\}) + c(\{4\})$$

$$h(\{1\}) = c(\{2,4\}) + c(\{3\})$$

$$h(\{1\}) = c(\{3,4\}) + c(\{2\})$$

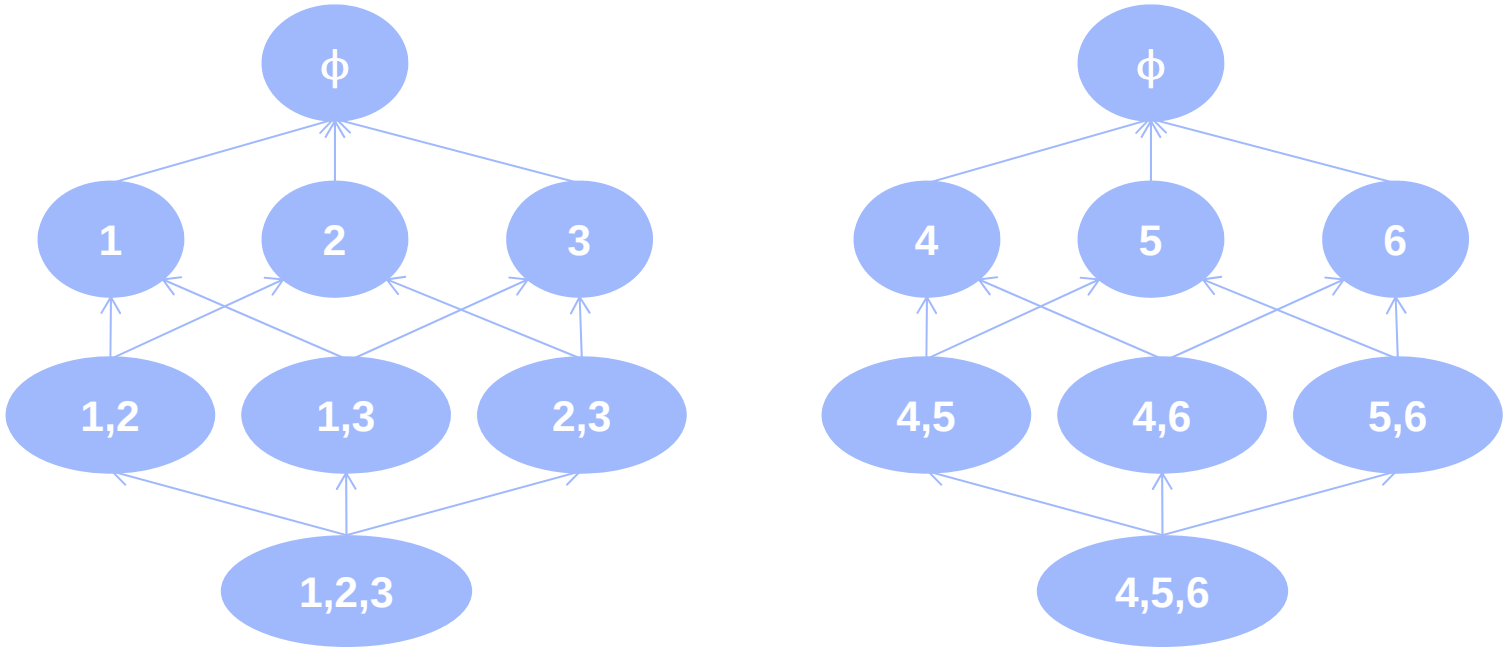
?

- **Can be formulated as a maximum-weight matching problem**
  - **$O(N^3)$ for $k=2$** [Papadimitriou&Steiglitz 1982]
  - **NP-hard for $k \geq 3$** [Garey&Johnson 1979]
- **Greedy method**
  - **Order the patterns based on amount of improvement**
  - **Greedily find the best available pattern at each step**

[Yuan, Malone, UAI-12]

Statically k-cycle conflict heuristic

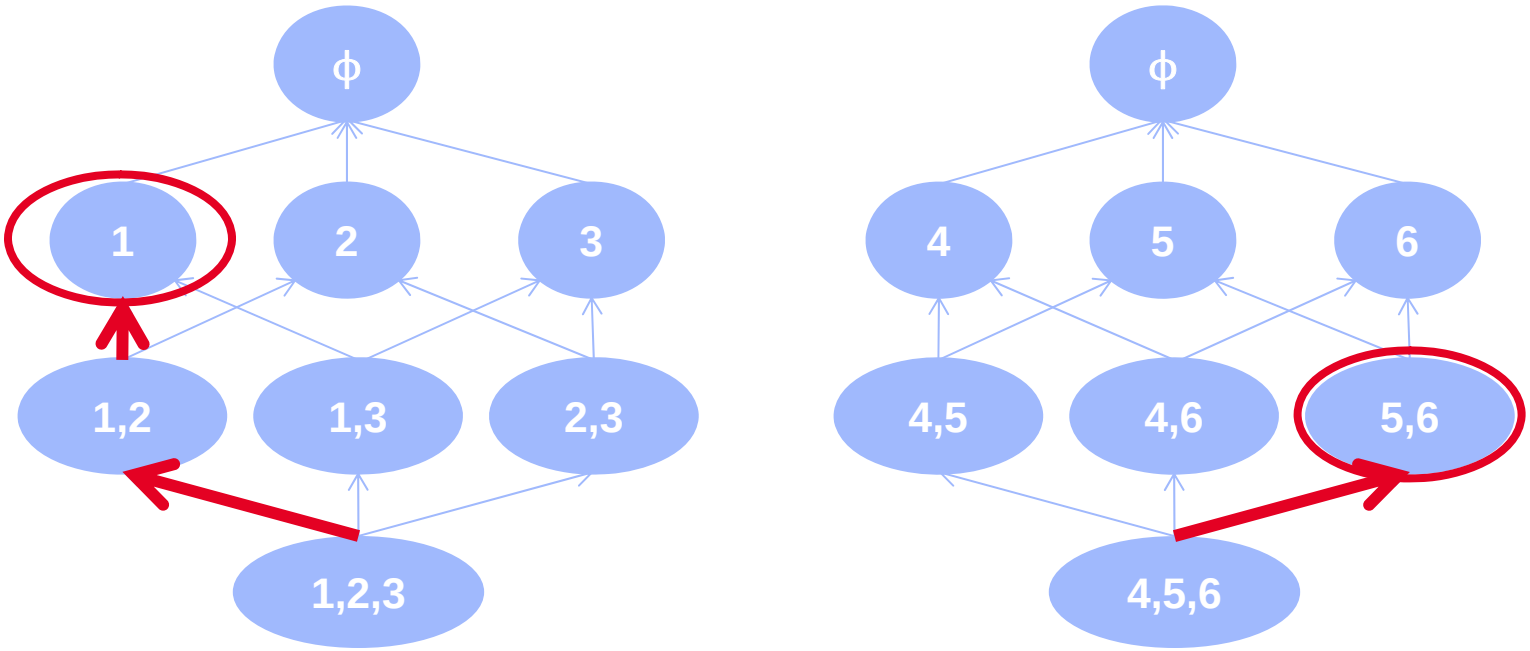
- **Calculate full pattern databases for static exclusive groups:**
  $\{1,2,3,4,5,6\} \Rightarrow \{1,2,3\}, \{4,5,6\}$



[Yuan, Malone, UAI-12]

Computing heuristic value using static PD

- **Sum costs of pattern databases according to static grouping**



$$h(\{1,5,6\}) = c(\{2,3\}) + c(\{4\}) = g^r(\{1\}) + g^r(\{5,6\})$$

[Yuan, Malone, UAI-12]

Properties of *k*-cycle conflict heuristic

- **Theorem: The *k*-cycle conflict heuristic is admissible**
  - **Dynamic version**
  - **Static version**
- **Theorem: The static *k*-cycle conflict heuristic is consistent**
- **Theorem: The dynamic *k*-cycle conflict heuristic is consistent**
  - **Under the condition that the maximum weight matching problems are solved optimally**
  - **Will lose the consistency property if using an approximation algorithm**

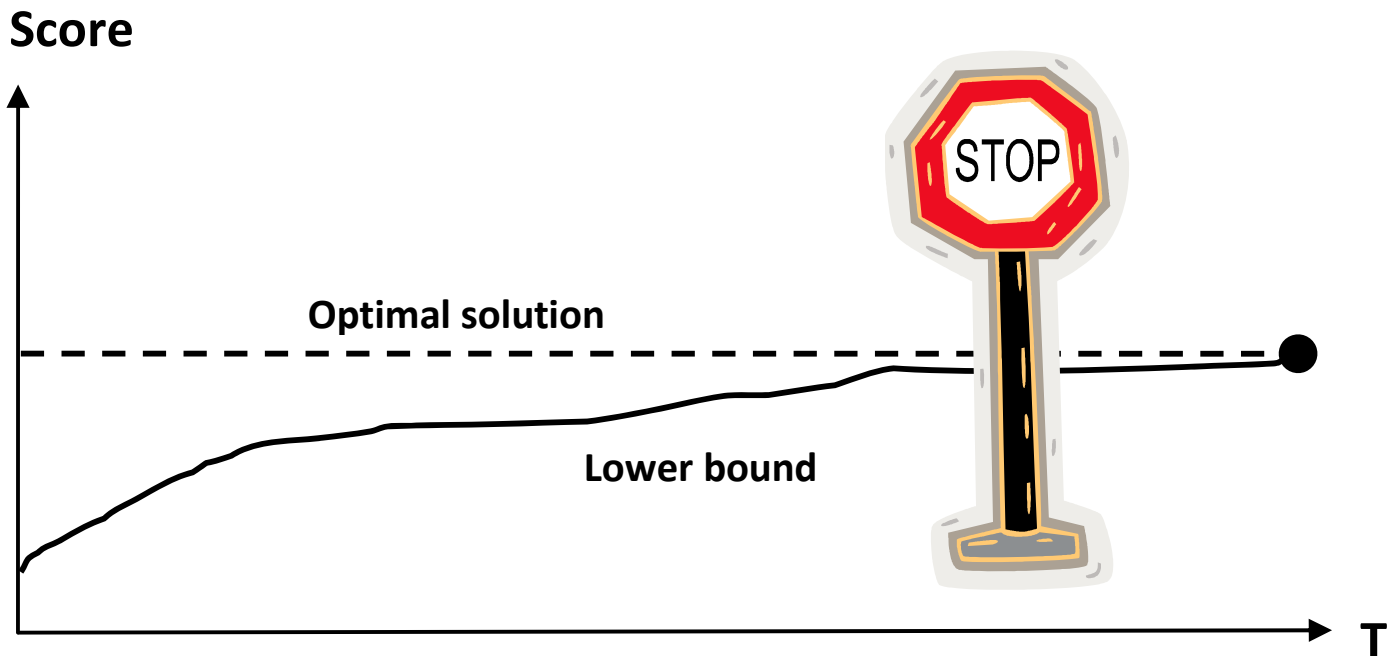
[Yuan, Malone, UAI-12]

Limitation of A* and BFBnB

- A* and BFBnB do not find any solution before the search finishes
- That means it may take a long time before
  - finding the optimal solution
  - or failing to find any solution
- Algorithms with anytime behavior are desirable
  - Find a solution quickly, and keep improving the solution until converging to an optimal solution if given enough resources
  - If stopped early, the algorithm can simply output the best solution so far
  - Have guaranteed error bounds

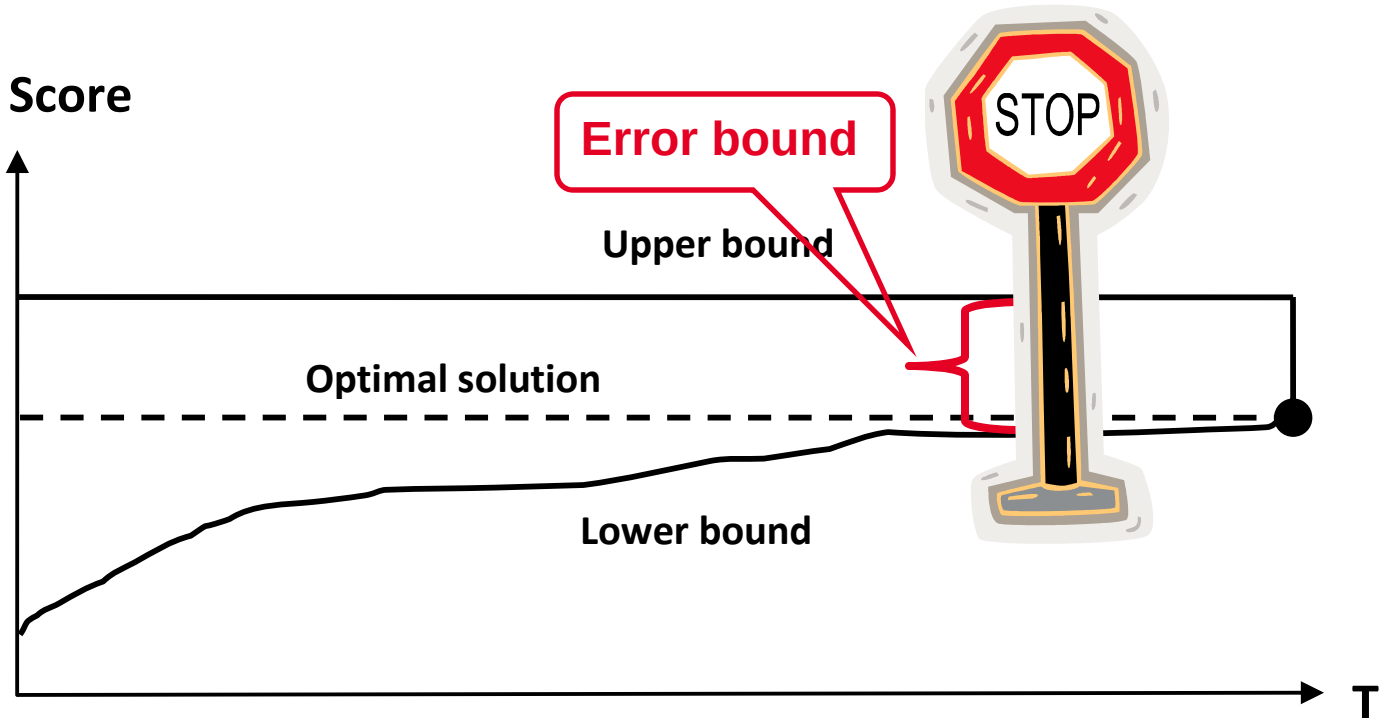
Error bounds: A*

- Lower bound: the least f-cost in the open list
- Upper bound: None
- Hence, A* either finds the optimal solution, or no solution at all



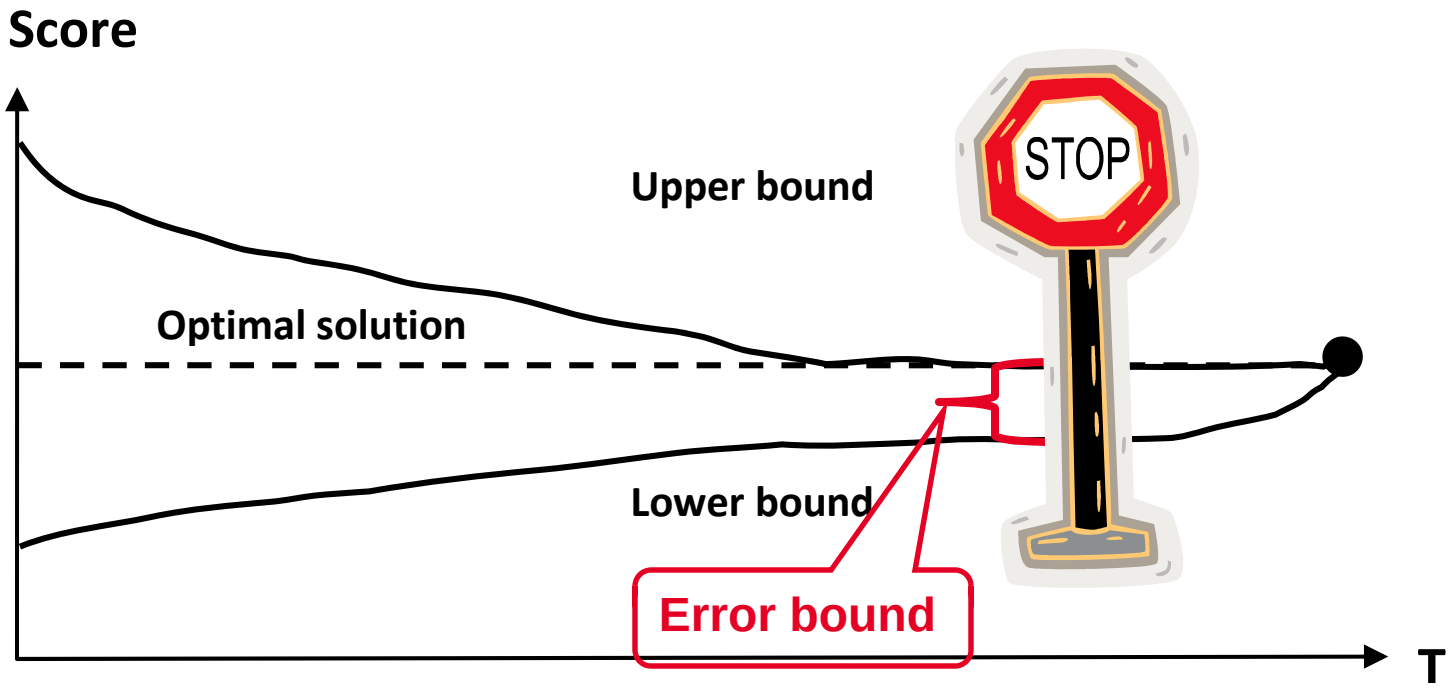
Error bounds: BFBnB

- Lower bound: the least f-cost in the last two layers
- Upper bound: the initial greedy solution



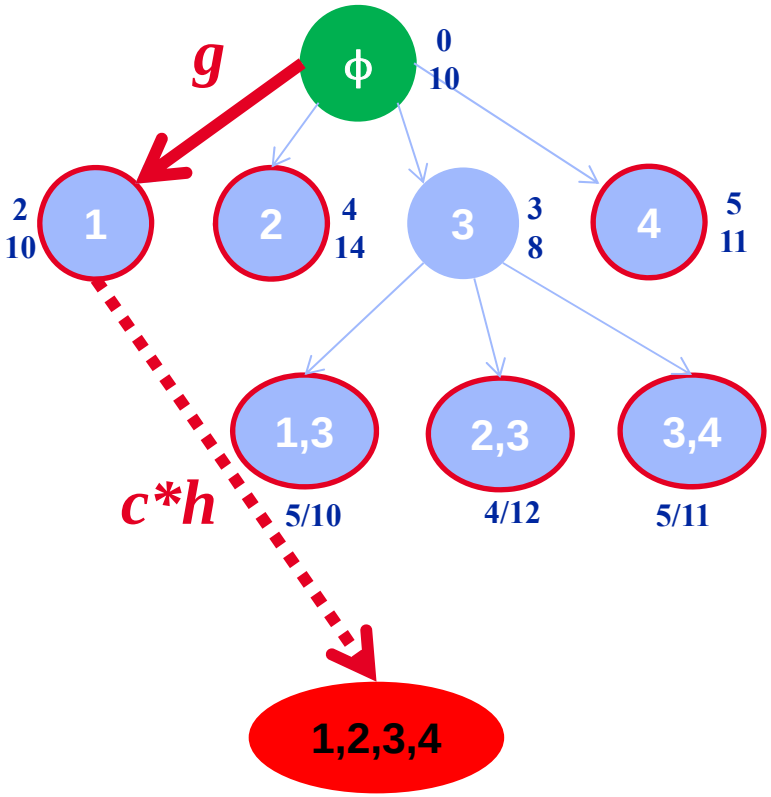
Error bounds: Anytime algorithms

- Lower bound: the least f-cost in the search frontier
- Upper bound: the current best solution so far



Anytime weighted A*

- **Weight the heuristic value *h* by a constant *c>=1.0***
  - **Make smaller *h* look more promising**
- **Perform best-first search until open list is empty**

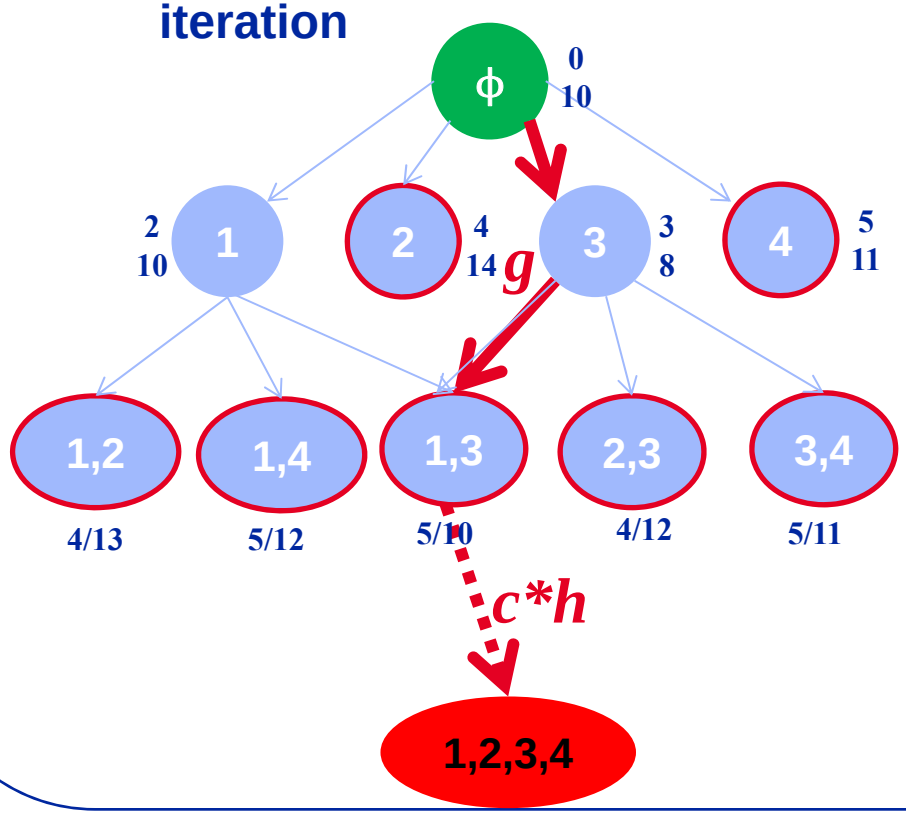


$f=g+c*h$

[Malone, Yuan UAI-13]

Anytime repairing A*

- **Weight the *h* by a constant *c>=1.0***
- **Start with a large weight c_0, and gradually decrease to *1.0***
  - **Perform A* search at each iteration**
  - **When finding better paths, delay reexpansions until the next iteration**

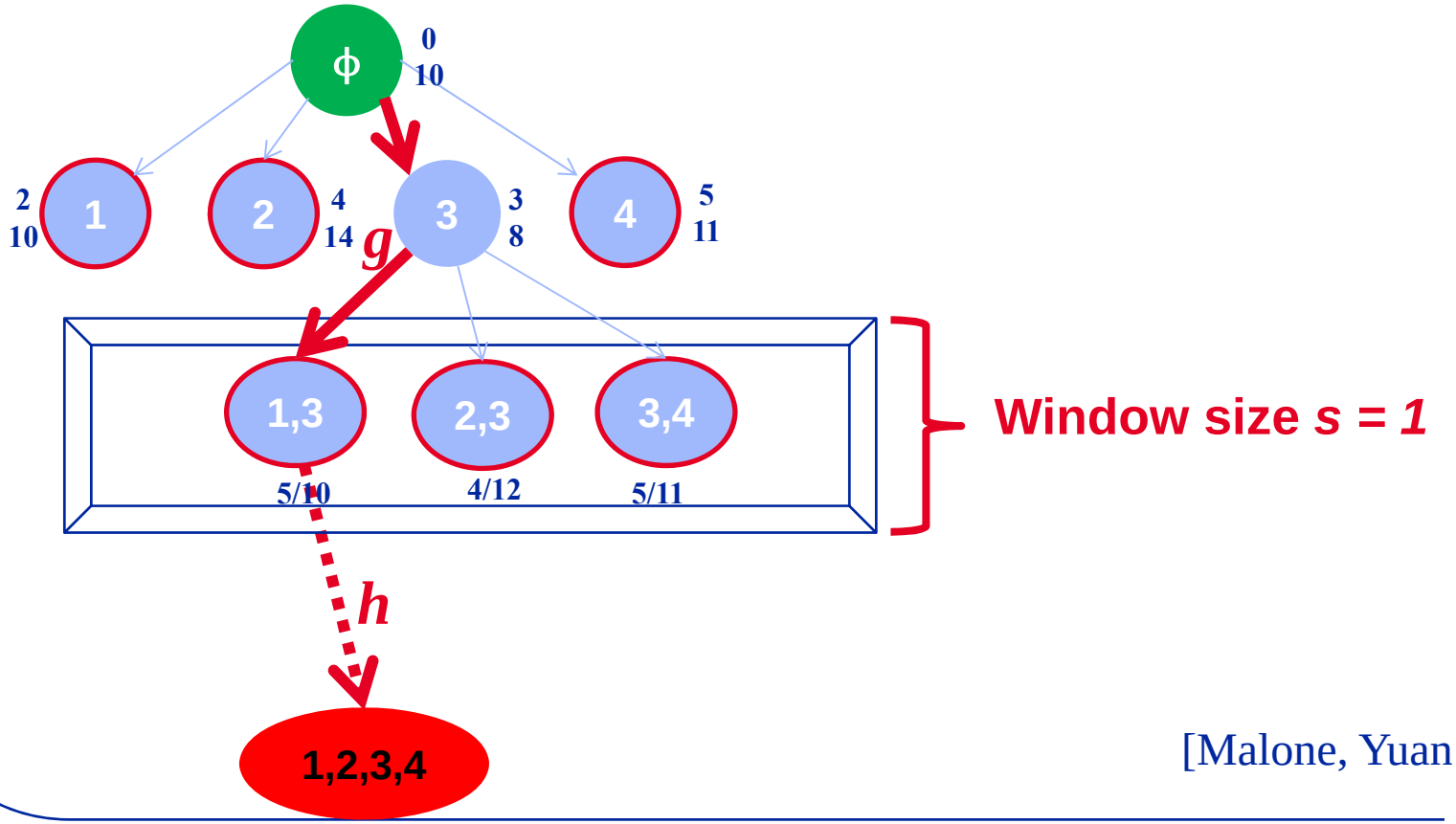


$$f=g+c*h$$

[Malone, Yuan UAI-13]

Anytime window A*

- **Use a window size *s* to limit the open list**
  - **Only include nodes within *s* steps from the deepest open node**
- **For each *s* from *1* to *total depth***
  - **Perform A* search with limited open list**



[Malone, Yuan UAI-13]

Outline

- Basics of Bayesian networks (30 minutes)
- Scoring functions (30 minutes)
- Dynamic programming (30 minutes)
- Admissible heuristic search (45 minutes)
- ➢ Integer linear programming (60 minutes)
- Empirical evaluations (15 minutes)

Integer Programming for Bayesian Network Structure Learning

James Cussens

University of York

IJCAI, 2013-08-05

The Belgian diet problem

	Fat	Sugar	Salt	Cost
Chocolate	4	6	1	5
Chips	6	1	8	4
Needs	12	8	4	

Minimise $5x + 4y$, subject to:

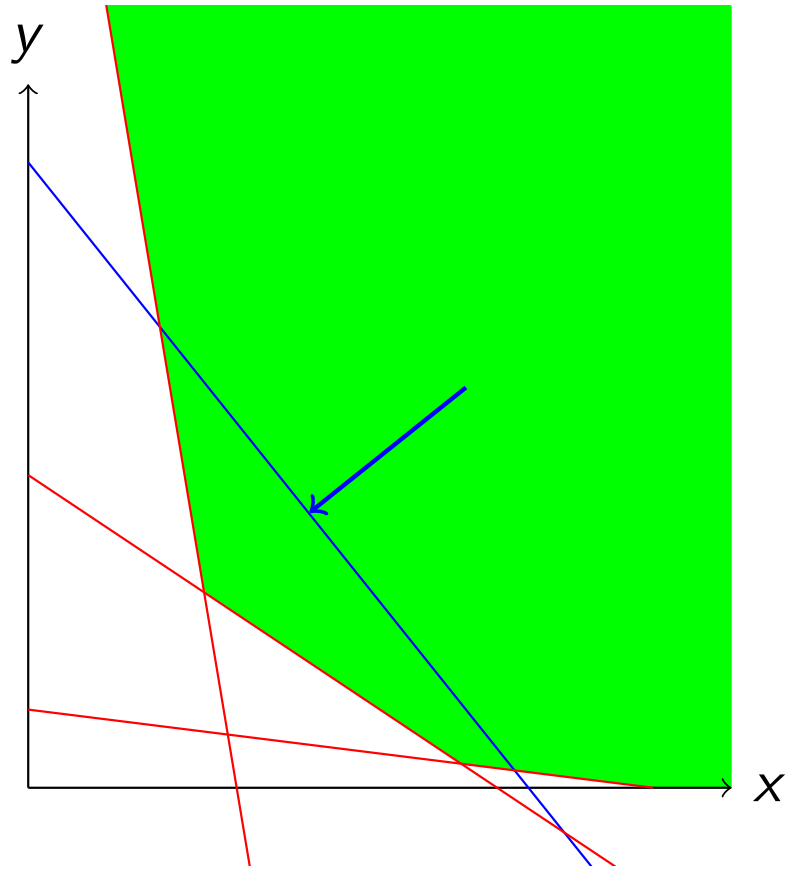
$x, y \geq 0$

$4x + 6y \geq 12$

$5x + y \geq 8$

$x + 8y \geq 4$

$x, y \in \mathbb{R}$



Solving an LP using SCIP

```
presolved problem has 2 variables
(0 bin, 0 int, 0 impl, 2 cont) and 3 constraints

LP iter|cols |rows |  dualbound | primalbound|  gap
     0 |   0 |   0 |        --  | 5.2000e+01 |    Inf
     2 |   2 |   3 | 1.0625e+01 | 5.2000e+01 | 389.41%
     2 |   2 |   3 | 1.0625e+01 | 1.0625e+01 |   0.00%

chocolate 1.125
chips 1.25
```

Discrete dieting

	Fat	Sugar	Salt	Cost
Chocolate	4	6	1	5
Chips	6	1	8	3
Needs	12	8	4	

Minimise $5x + 4y$, subject to:

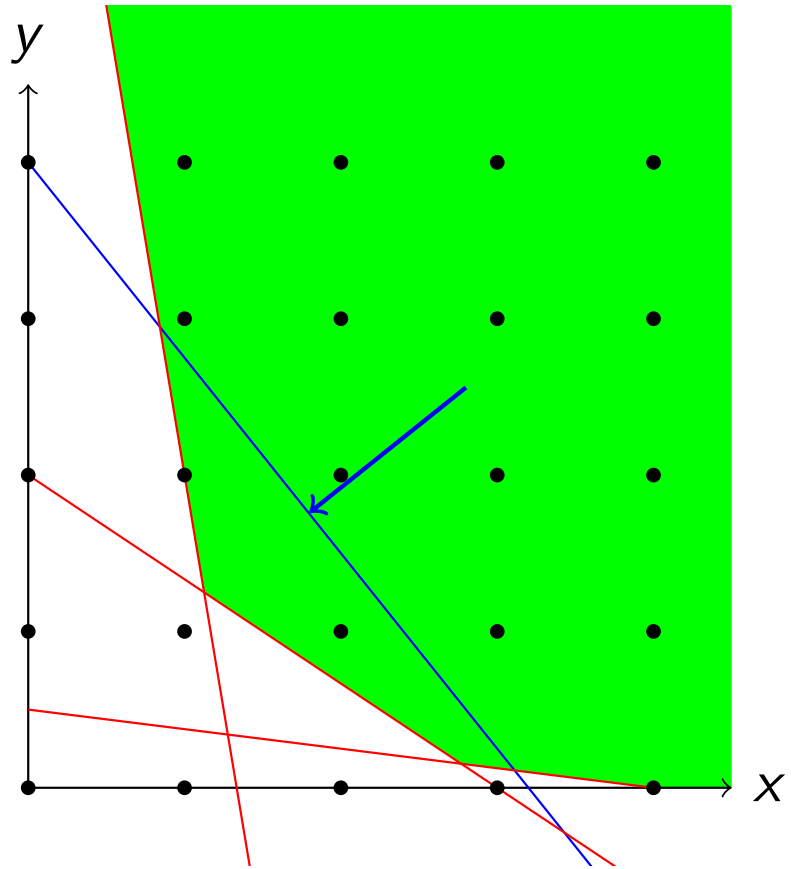
$x, y \geq 0$

$4x + 6y \geq 12$

$5x + y \geq 8$

$x + 8y \geq 4$

$x, y \in \mathbb{Z}$



Discrete dieting — Finding a solution

	Fat	Sugar	Salt	Cost
Chocolate	4	6	1	5
Chips	6	1	8	3
Needs	12	8	4	

Minimise $5x + 4y$, subject to:

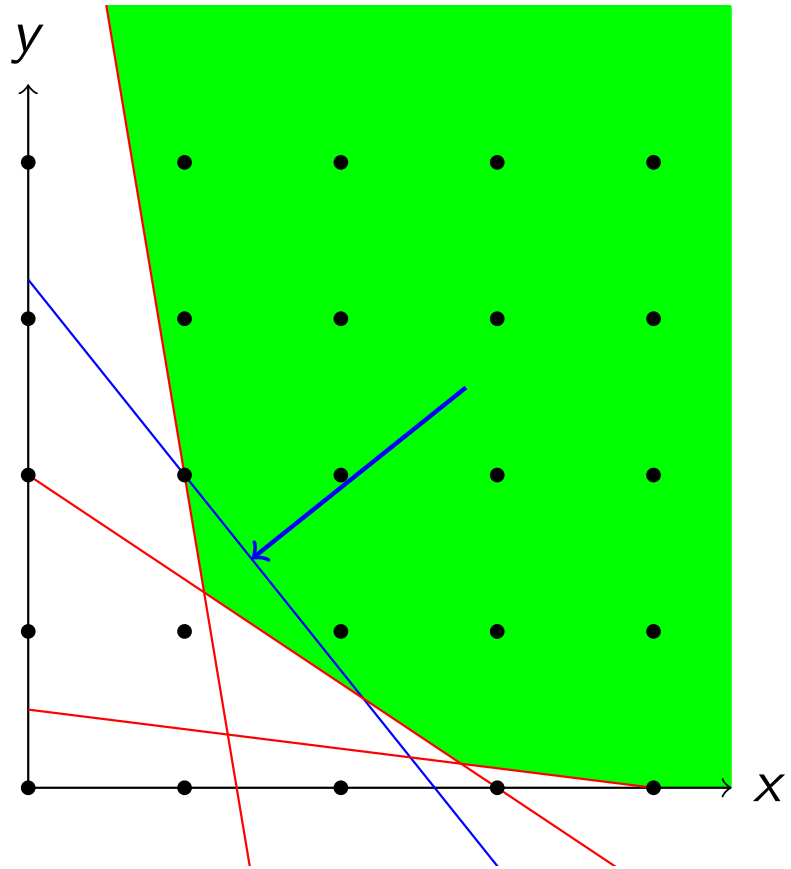
$x, y \geq 0$

$4x + 6y \geq 12$

$5x + y \geq 8$

$x + 8y \geq 4$

$x, y \in \mathbb{Z}$



Discrete dieting — Finding the solution

	Fat	Sugar	Salt	Cost
Chocolate	4	6	1	5
Chips	6	1	8	3
Needs	12	8	4	

Minimise $5x + 4y$, subject to:

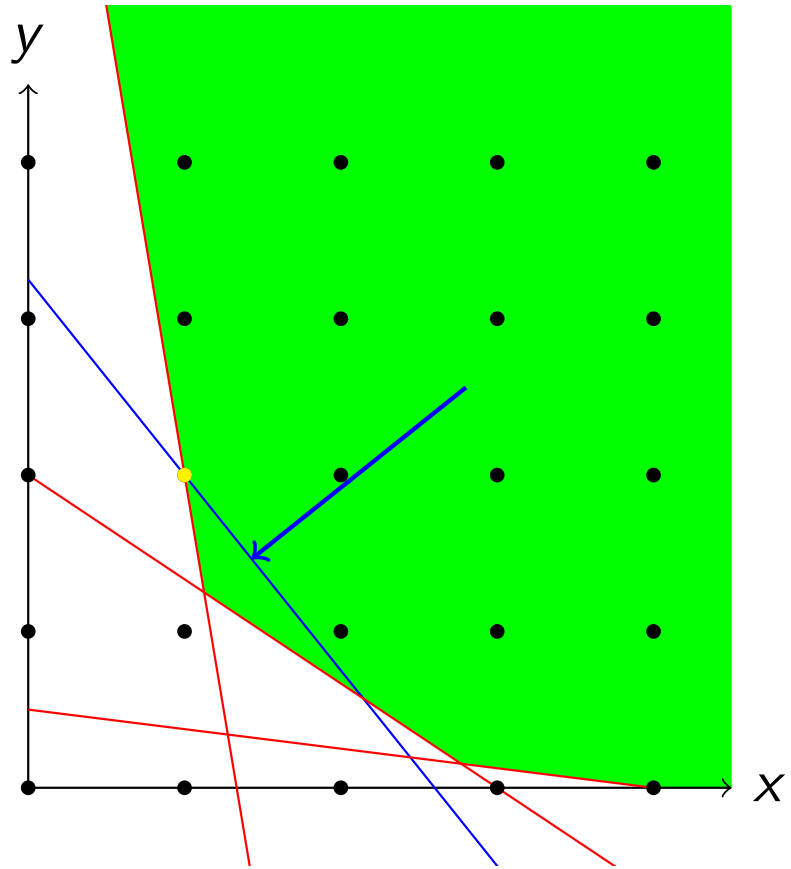
$x, y \geq 0$

$4x + 6y \geq 12$

$5x + y \geq 8$

$x + 8y \geq 4$

$x, y \in \mathbb{Z}$

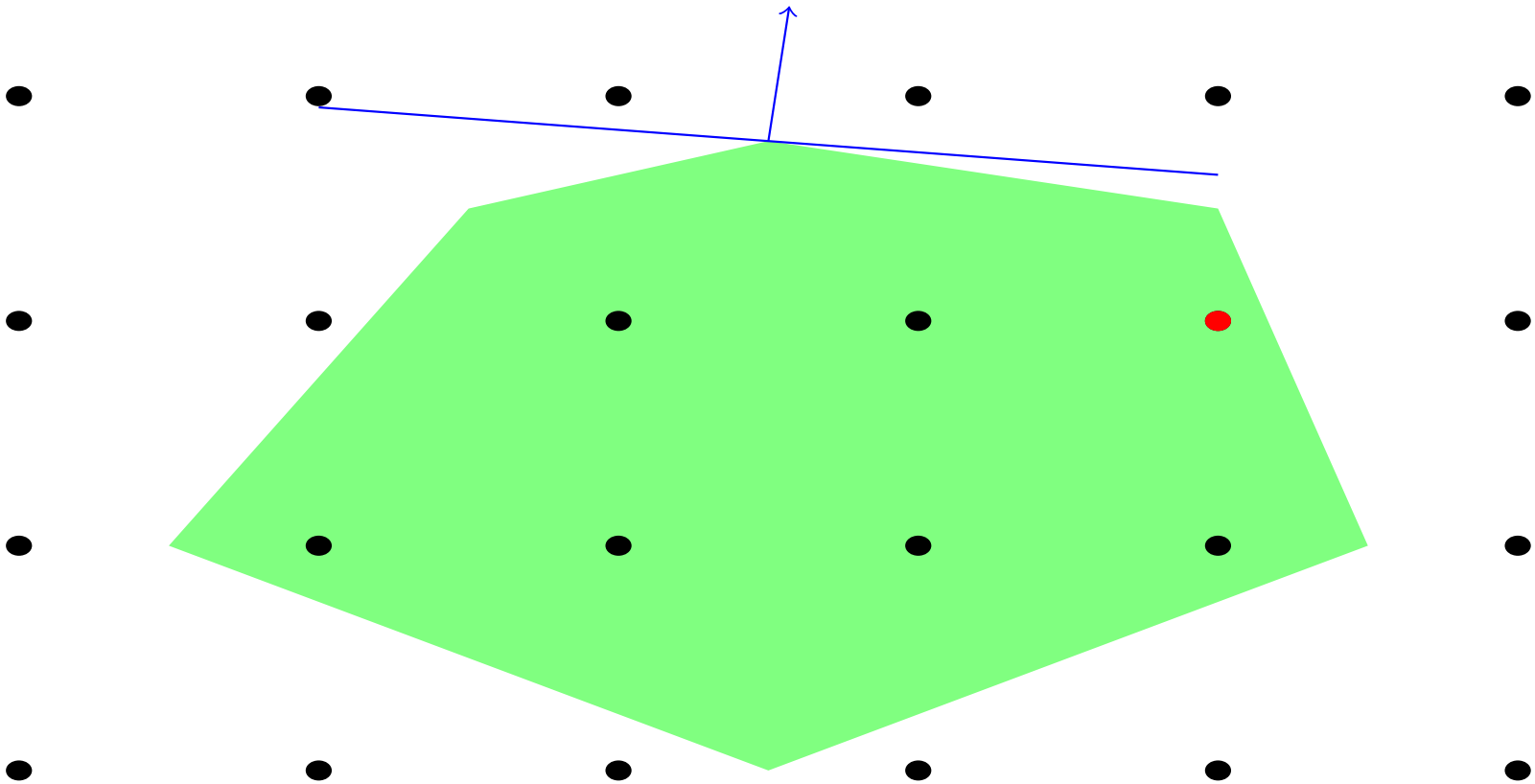


Solving an IP using SCIP

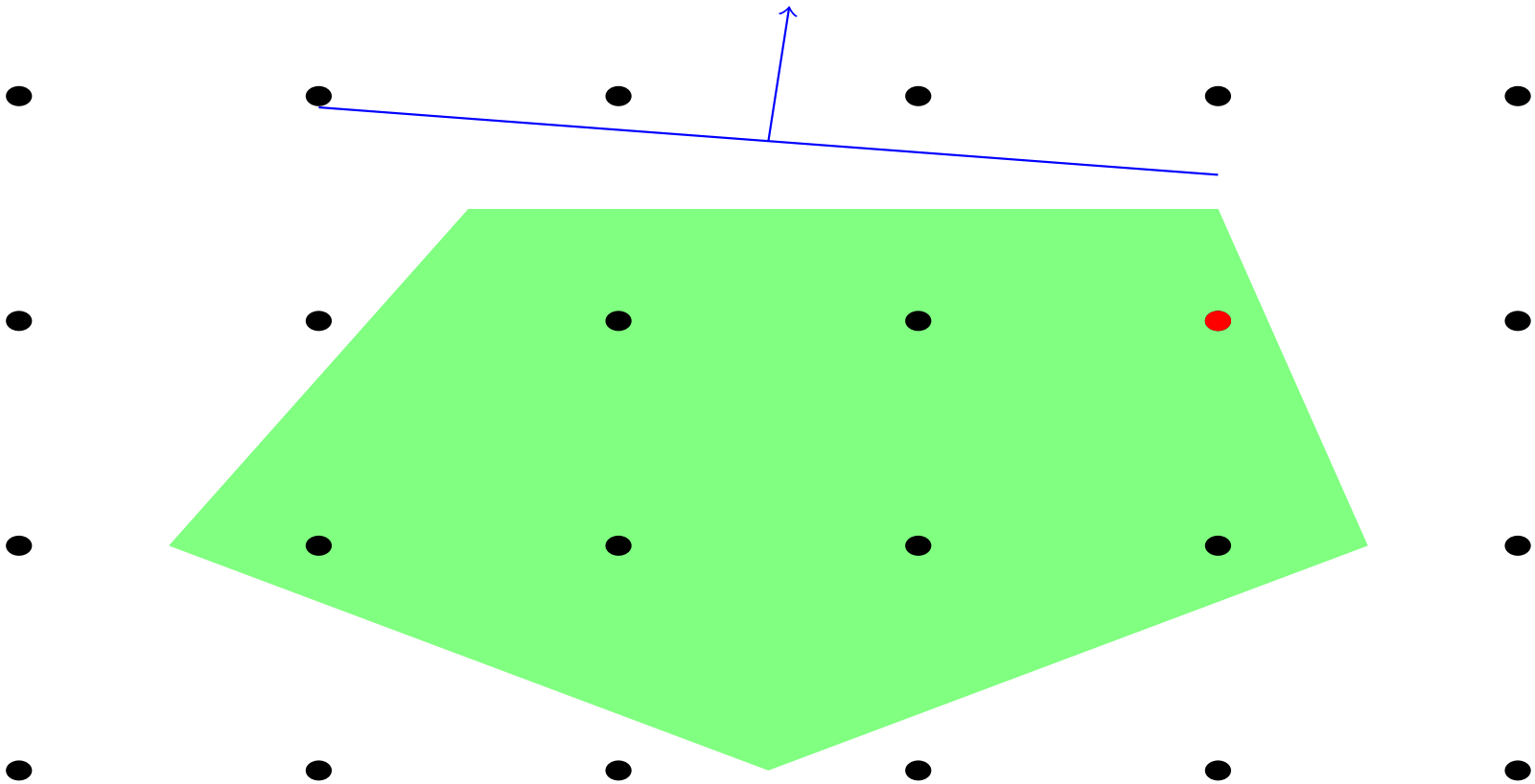
```
presolved problem has 2 variables
(0 bin, 2 int, 0 impl, 0 cont) and 3 constraints

LP iters|cols |rows |cuts |  dualbound | p'bnd    |  gap
      0  |   0 |   0 |   0 |        --  | 5.2e+01 |   Inf
      0  |   2 |   3 |   0 |        --  | 2.0e+01 |   Inf
      2  |   2 |   3 |   0 | 1.0625e+01 | 2.0e+01 | 88.24%
      2  |   2 |   3 |   0 | 1.0625e+01 | 1.8e+01 | 69.41%
      2  |   2 |   3 |   0 | 1.0625e+01 | 1.3e+01 | 22.35%
      3  |   2 |   4 |   1 | 1.3000e+01 | 1.3e+01 |  0.00%
      3  |   2 |   4 |   1 | 1.3000e+01 | 1.3e+01 |  0.00%
```

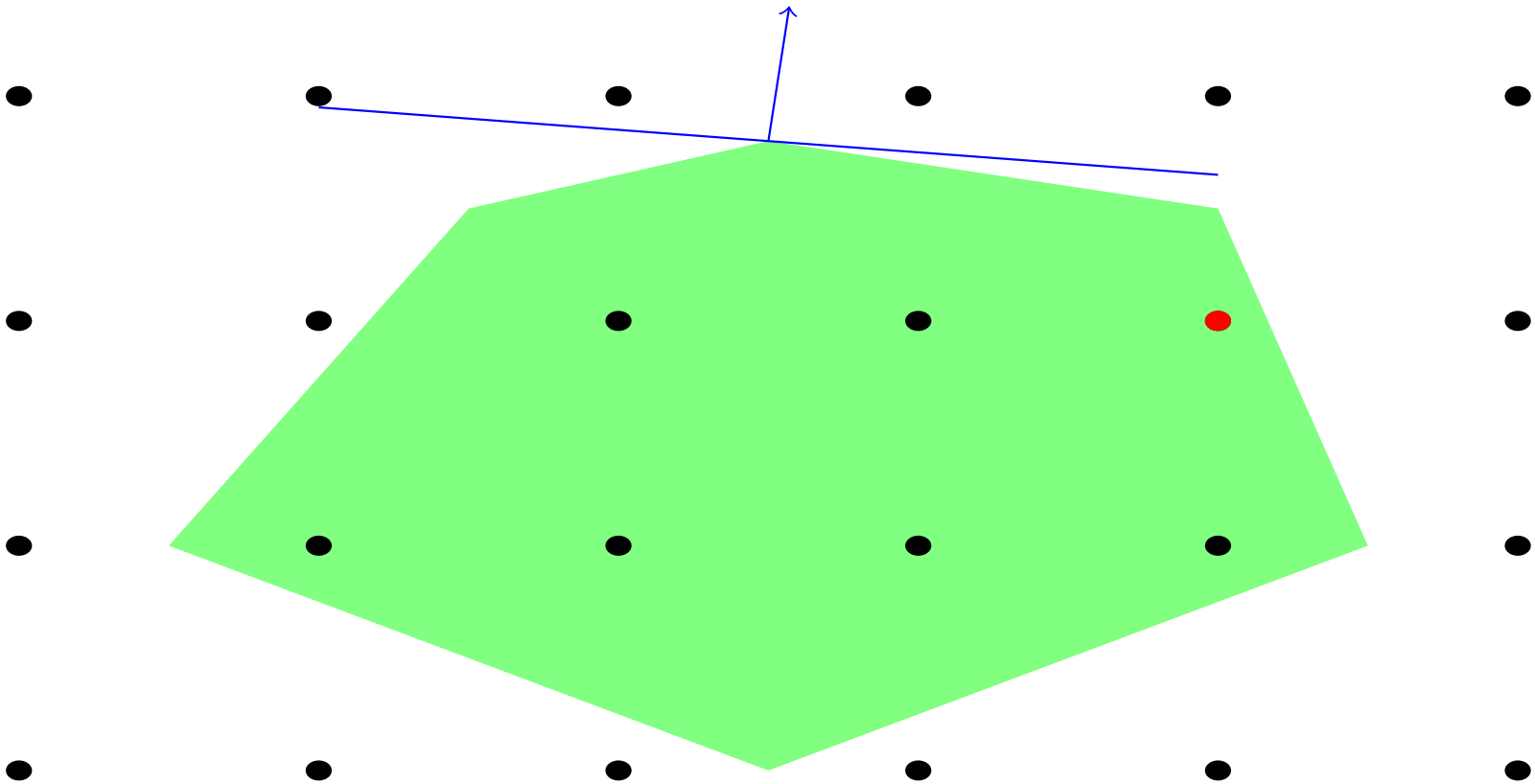
Cutting planes



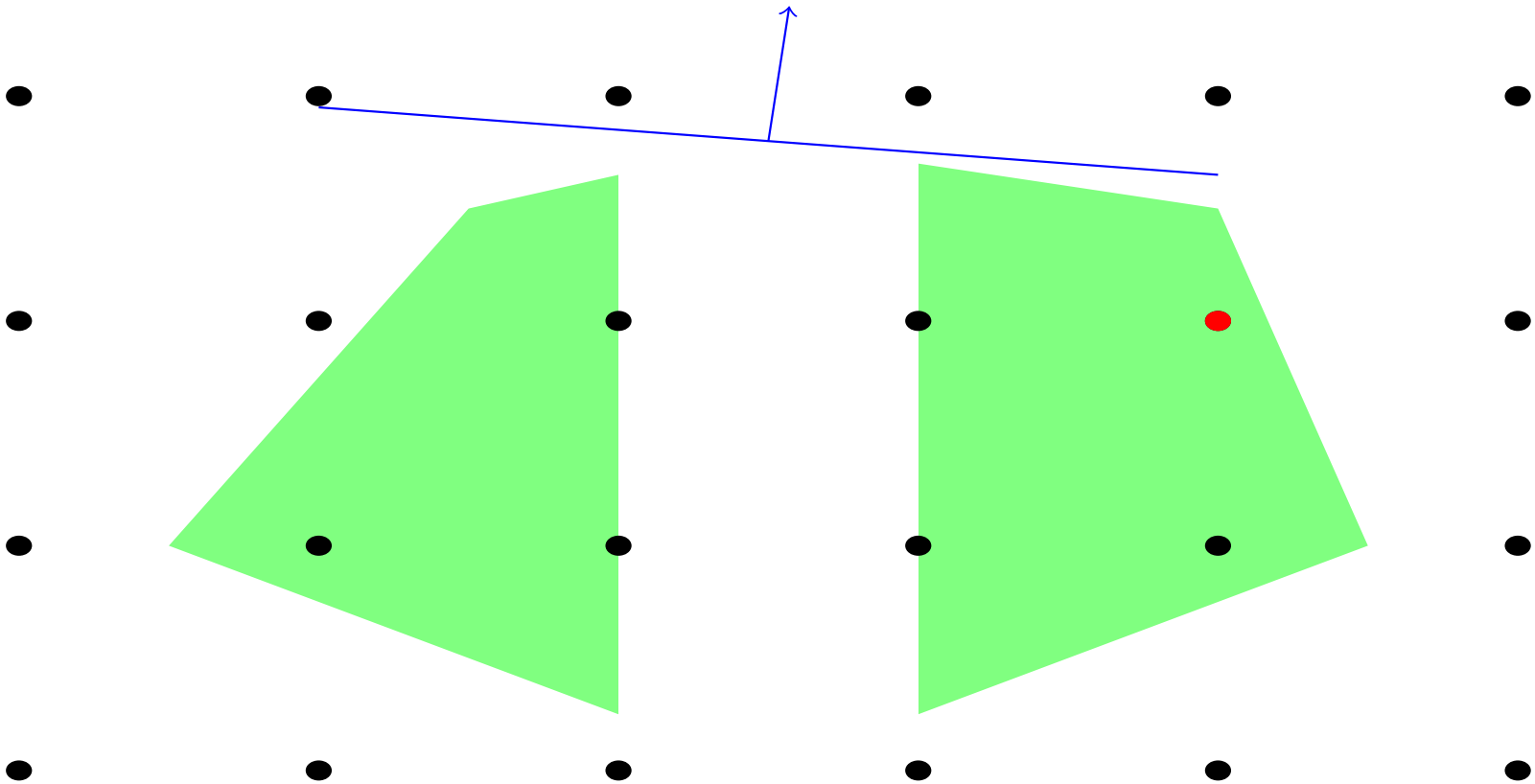
Cutting planes



Branch-and-bound



Branch-and-bound



(Mixed) Integer (linear) programming

- All variables x_i have numeric values.
- Binary (0/1), Integer and Real
- **I**LP: Linear objective function $\sum c_i x_i$
- Pure **I**LP: All constraints are linear.

Solving the *linear relaxation* (where integrality restrictions are removed) provides a crucial upper bound (when maximising).

Branch and cut algorithm

Two problems:
(a) finding an optimal solution
(b) proving it's optimal

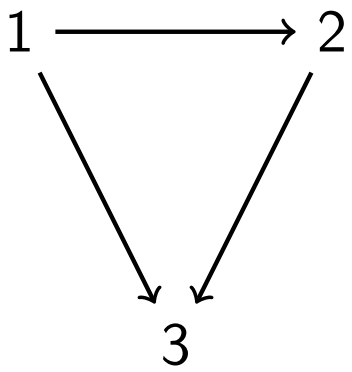
Solving the LP relaxation is fast. So ...

```
1. Let x* be the LP solution
2. If there are valid linear inequalities
        not satisfied by x*
        add them (as cutting planes) and go to 1.
   Else if x* is integer-valued then
        the current problem is solved
   Else branch on a variable with
        non-integer value in x*
        to create two new sub-IPs.
```

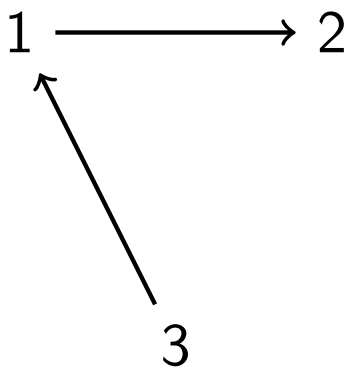
Encoding graphs with binary 'family' variables

- Suppose there are p fully-observed discrete variables V in some dataset. We want to find a MAP BN (with p vertices) conditional on this data.
- Can encode any graph by creating a binary variable $I(u \leftarrow W)$ for each BN variable $u \in V$ and each candidate parent set W.

Encoding DAGs



$$I(1 \leftarrow \emptyset) = 1,\ I(2 \leftarrow \{1\}) = 1,\ I(3 \leftarrow \{1,2\}) = 1$$



$$I(1 \leftarrow \{3\}) = 1,\ I(2 \leftarrow \{1\}) = 1,\ I(3 \leftarrow \emptyset) = 1$$

Encoding graphs with binary 'family' variables

- Suppose there are p fully-observed discrete variables V in some dataset. We want to find a MAP BN (with p vertices) conditional on this data.
- Can encode any graph by creating a binary variable $I(u \leftarrow W)$ for each BN variable $u \in V$ and each candidate parent set W.
- Assume known parameters (pedigrees) or Dirichlet parameter priors (general BN) and a uniform (or at least 'decomposable') structural prior.
- Each $I(u \leftarrow W)$ has a *local score* $c(u, W)$.

Instantiate the $I(u \leftarrow W)$ to maximise:
$\sum_{u,W} c(u, W) I(u \leftarrow W)$
subject to the $I(u \leftarrow W)$ representing a DAG.

Ruling out non-DAGs with linear constraints

$$\forall u \in V : \sum_{W} I(u \leftarrow W) = 1$$

$$\text{Where } C \subseteq V : \sum_{u \in C} \sum_{W : W \cap C = \emptyset} I(u \leftarrow W) \geq 1 \qquad (1)$$

- Let x^* be the solution to the LP relaxation. We search for a cluster C such that x^* violates (1) and then add (1) to get a new LP.
- Repeat as long as a *cutting plane* can be found.
- These constraints introduced by Jaakkola et al, AISTATS2010 [Jaakkola et al., 2010].

Linear inequalities for 3-node DAGs (set covering)

$$I(1 \leftarrow \emptyset) + I(1 \leftarrow \{2\}) + I(1 \leftarrow \{3\}) + I(1 \leftarrow \{2,3\}) = 1 \quad (2)$$

$$I(2 \leftarrow \emptyset) + I(2 \leftarrow \{1\}) + I(2 \leftarrow \{3\}) + I(2 \leftarrow \{1,3\}) = 1 \quad (3)$$

$$I(3 \leftarrow \emptyset) + I(3 \leftarrow \{1\}) + I(3 \leftarrow \{2\}) + I(3 \leftarrow \{1,2\}) = 1 \quad (4)$$

$$I(1 \leftarrow \emptyset) + I(1 \leftarrow \{3\}) + I(2 \leftarrow \emptyset) + I(2 \leftarrow \{3\}) \geq 1 \quad (5)$$

$$I(1 \leftarrow \emptyset) + I(1 \leftarrow \{2\}) + I(3 \leftarrow \emptyset) + I(3 \leftarrow \{2\}) \geq 1 \quad (6)$$

$$I(2 \leftarrow \emptyset) + I(2 \leftarrow \{1\}) + I(3 \leftarrow \emptyset) + I(3 \leftarrow \{1\}) \geq 1 \quad (7)$$

$$I(1 \leftarrow \emptyset) + I(2 \leftarrow \emptyset) + I(3 \leftarrow \emptyset) \geq 1 \quad (8)$$

Linear inequalities for 3-node DAGs (knapsack)

$$I(1 \leftarrow \emptyset) + I(1 \leftarrow \{2\}) + I(1 \leftarrow \{3\}) + I(1 \leftarrow \{2,3\}) = 1 \quad (9)$$

$$I(2 \leftarrow \emptyset) + I(2 \leftarrow \{1\}) + I(2 \leftarrow \{3\}) + I(2 \leftarrow \{1,3\}) = 1 \quad (10)$$

$$I(3 \leftarrow \emptyset) + I(3 \leftarrow \{1\}) + I(3 \leftarrow \{2\}) + I(3 \leftarrow \{1,2\}) = 1 \quad (11)$$

$$I(1 \leftarrow \{2\}) + I(1 \leftarrow \{2,3\}) + I(2 \leftarrow \{1\}) + I(2 \leftarrow \{1,3\}) \leq 1 \quad (12)$$

$$I(1 \leftarrow \{3\}) + I(1 \leftarrow \{2,3\}) + I(3 \leftarrow \{1\}) + I(3 \leftarrow \{1,2\}) \leq 1 \quad (13)$$

$$I(3 \leftarrow \{2\}) + I(3 \leftarrow \{1,2\}) + I(3 \leftarrow \{2\}) + I(3 \leftarrow \{1,2\}) \leq 1 \quad (14)$$

$$\begin{aligned} & I(1 \leftarrow \{2\}) + I(1 \leftarrow \{3\}) + I(1 \leftarrow \{2,3\}) \\ & + I(2 \leftarrow \{1\}) + I(2 \leftarrow \{3\}) + I(2 \leftarrow \{1,3\}) \\ & + I(3 \leftarrow \{1\}) + I(3 \leftarrow \{2\}) + I(1 \leftarrow \{1,2\}) \leq 2 \end{aligned} \quad (15)$$

Time for a demo

Do:

- Pigs_100_1_2
- alarm

Primal heuristics in IP

- A good (typically suboptimal) solution helps prune the search tree.
- Can also help in the root search node due to *reduced cost strengthening*.
- If we fail to solve to optimality even more important to have a reasonable solution.

Sink finding

- As we learn from [Silander and Myllymäki, 2006] ...
- Every DAG has at least one sink node (node with no children).
- For this node we can choose the 'best' parents without fear of creating a cycle.
- Once a sink v_p selected from V then just worry about learning the best BN with nodes $V \setminus \{v_p\}$.
- Basically the same as finding the best total order (in reverse).

Using the LP solution to find sinks

$I(1 \leftarrow W_{1,1})$	$I(1 \leftarrow W_{1,2})$	$\ldots$	$I(1 \leftarrow W_{1,k_1})$
$I(2 \leftarrow W_{2,1})$	$I(2 \leftarrow W_{2,2})$	$\ldots$	$I(2 \leftarrow W_{2,k_2})$
$I(3 \leftarrow W_{3,1})$	$I(3 \leftarrow W_{3,2})$	$\ldots$	$I(3 \leftarrow W_{3,k_3})$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$I(p \leftarrow W_{p,1})$	$I(p \leftarrow W_{p,2})$	$\ldots$	$I(p \leftarrow W_{p,k_p})$

- For each variable, order its parent set choices from best to worst.

Using the LP solution to find sinks

$I(1 \leftarrow W_{1,1})$	$I(1 \leftarrow W_{1,2})$	$\ldots$	$I(1 \leftarrow W_{1,k_1})$
$I(2 \leftarrow W_{2,1})$	$I(2 \leftarrow W_{2,2})$	$\ldots$	$I(2 \leftarrow W_{2,k_2})$
$I(3 \leftarrow W_{3,1})$	$I(3 \leftarrow W_{3,2})$	$\ldots$	$I(3 \leftarrow W_{3,k_3})$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$I(p \leftarrow W_{p,1})$	$I(p \leftarrow W_{p,2})$	$\ldots$	$I(p \leftarrow W_{p,k_p})$

- (With only the acyclicity constraint) for an optimal BN at least one BN variable has its best parent set selected.

Using the LP solution to find sinks

$I(1 \leftarrow W_{1,1})$	$I(1 \leftarrow W_{1,2})$	...	$I(1 \leftarrow W_{1,k_1})$
$\mathbf{I(2 \leftarrow W_{2,1})}$	$I(2 \leftarrow W_{2,2})$	...	$I(2 \leftarrow W_{2,k_2})$
$I(3 \leftarrow W_{3,1})$	$I(3 \leftarrow W_{3,2})$	...	$I(3 \leftarrow W_{3,k_3})$
...	...	...	...
$I(p \leftarrow W_{p,1})$	$I(p \leftarrow W_{p,2})$	...	$I(p \leftarrow W_{p,k_p})$

- (With only the acyclicity constraint) for an optimal BN at least one BN variable has its best parent set selected.
- Let x^* be the LP solution and suppose $x^*_{I(2 \leftarrow W_{2,1})}$ is closer to 1 than the best parent set choice for any other variable.

Using the LP solution to find sinks

$I(1 \leftarrow W_{1,1})$	$I(1 \leftarrow W_{1,2})$	$\ldots$	$I(1 \leftarrow W_{1,k_1})$
$\mathbf{I(2 \leftarrow W_{2,1})}$	~~$I(2 \leftarrow W_{2,2})$~~	~~$\ldots$~~	~~$I(2 \leftarrow W_{2,k_2})$~~
$I(3 \leftarrow W_{3,1})$	$I(3 \leftarrow W_{3,2})$	$\ldots$	$I(3 \leftarrow W_{3,k_3})$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$I(p \leftarrow W_{p,1})$	$I(p \leftarrow W_{p,2})$	$\ldots$	$I(p \leftarrow W_{p,k_p})$

- (With only the acyclicity constraint) for an optimal BN at least one BN variable has its best parent set selected.
- Let x^* be the LP solution and suppose $x^*_{I(2 \leftarrow W_{2,1})}$ is closer to 1 than the best parent set choice for any other variable.
- Select it.

Using the LP solution to find sinks

~~$I(1 \leftarrow W_{1,1})$~~	$I(1 \leftarrow W_{1,2})$	$\ldots$	$I(1 \leftarrow W_{1,k_1})$
$\mathbf{I(2 \leftarrow W_{2,1})}$	~~$I(2 \leftarrow W_{2,2})$~~	~~$\ldots$~~	~~$I(2 \leftarrow W_{2,k_2})$~~
$I(3 \leftarrow W_{3,1})$	~~$I(3 \leftarrow W_{3,2})$~~	$\ldots$	$I(3 \leftarrow W_{3,k_3})$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
~~$I(p \leftarrow W_{p,1})$~~	~~$I(p \leftarrow W_{p,2})$~~	$\ldots$	$I(p \leftarrow W_{p,k_p})$

- (With only the acyclicity constraint) for an optimal BN at least one BN variable has its best parent set selected.
- Let x^* be the LP solution and suppose $x^*_{I(2 \leftarrow W_{2,1})}$ is closer to 1 than the best parent set choice for any other variable.
- Select it.
- Suppose 2 is a member of $W_{1,1}$, $W_{3,2}$, $W_{p,1}$ and $W_{p,2}$

Sink finding primal heuristic

- The BN returned is always best for some total ordering.
- Basically a greedy search for such a BN 'near' the LP solution (L_1).
- Complications if some IP variables already fixed (due to branching).

Implied constraints

- The 'cluster' constraints of [Jaakkola et al., 2010] ensure that any integer solution is a DAG :-)
- But they do not define the convex hull of DAGs. :-(
- Adding in at least some of the *facets* of the convex hull (at least those which pass through an optimal DAG) can provide dramatic improvements.
- SCIP adds some extra cuts automatically (e.g. Gomory).

An extra facet for 3-node DAGs

This point:
$I(1 \leftarrow \{2,3\}) = 1/2, I(1 \leftarrow \{\emptyset\}) = 1/2$
$I(2 \leftarrow \{1,3\}) = 1/2, I(2 \leftarrow \{\emptyset\}) = 1/2$
$I(3 \leftarrow \{1,2\}) = 1/2, I(3 \leftarrow \{\emptyset\}) = 1/2$
is an extreme point of the polytope (in $\mathbb{R}^9$) defined by the 3 convexity constraints and 4 cluster constraints.
Need to add in:

$$I(1 \leftarrow \{2,3\}) + I(2 \leftarrow \{1,3\}) + I(3 \leftarrow \{1,2\}) \leq 1$$

to remove it (and get the convex hull of 3-node DAGs).
Generalising:

$$\forall C \subseteq V : \sum_{u \in C} \sum_{W : C \setminus \{u\} \subseteq W} I(u \leftarrow W) \leq 1 \qquad (16)$$

Effect of tightening the LP relaxation

- Adding in these extra constraints provides a dramatic speed-up, particularly for larger examples (hundreds of BN nodes).
- Currently experimenting with (generalisations of) facets of the 4-node DAG polytope.

Propagation

- If, say, $I(1 \leftarrow 2, 3)$ and $I(4 \leftarrow 1)$ set to 1 then immediately fix e.g. $I(2 \leftarrow 4)$ to 0.
- Compute transitive closure efficiently!

Conditional independence constraints

- Solutions only checked for conditional independence constraints after they get through integrality and acyclicity constraints.
- If a DAG does not meet a conditional independence constraint a cutting plane ruling out just that DAG is added.
- There is also some simple presolving and some obvious simple valid inequalities added at the start.
- Also in next release. (Do demo with citest.constraints)

Column generation

- Just as one can create constraints on the fly (cutting planes) one can also create variables dynamically (column generation).
- Think of the not-currently-created variables as being initially fixed to zero.
- After solving the LP we look for a variable with *negative reduced cost*. If we can't find one we have all the variables we need for an optimal solution.
- For reduced cost we need the objective coefficient of the new variable and dual values for all the constraints in which it will appear.

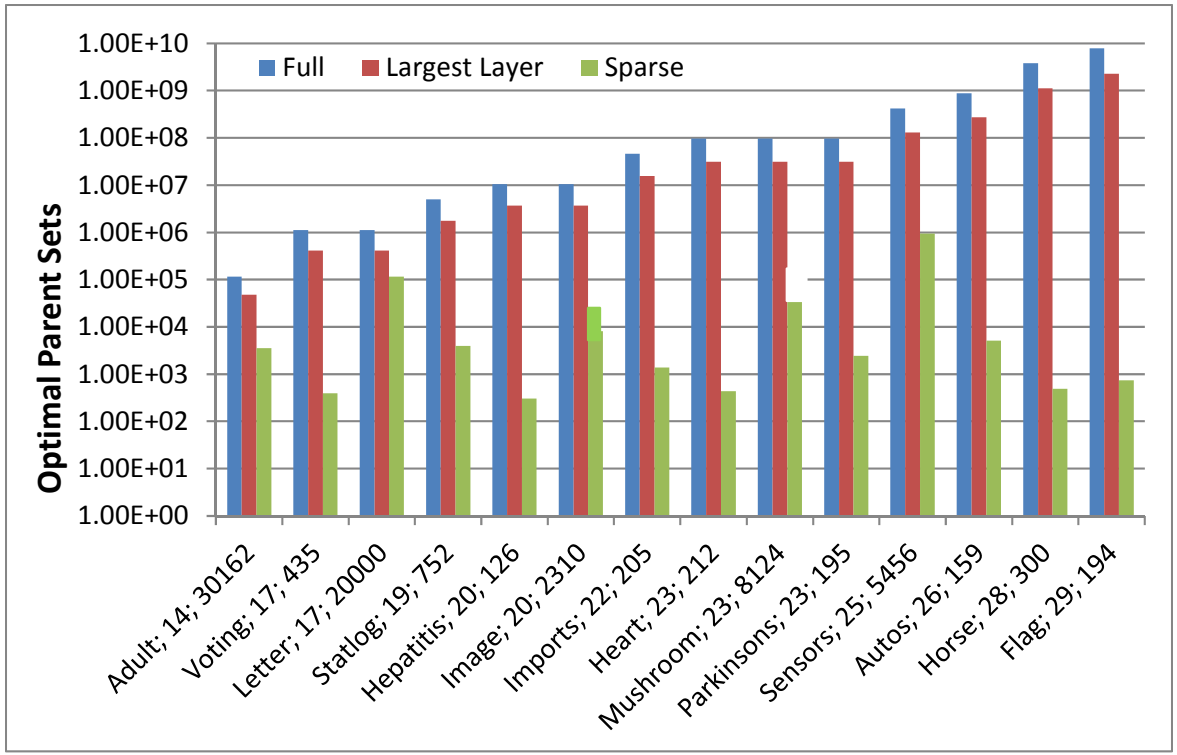

Jaakkola, T., Sontag, D., Globerson, A., and Meila, M. (2010).
Learning Bayesian network structure using LP relaxations.
In *Proceedings of 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 358–365. Journal of Machine Learning Research Workshop and Conference Proceedings.


Silander, T. and Myllymäki, P. (2006).
A simple approach for finding the globally optimal Bayesian network structure.
In *UAI*.

Outline

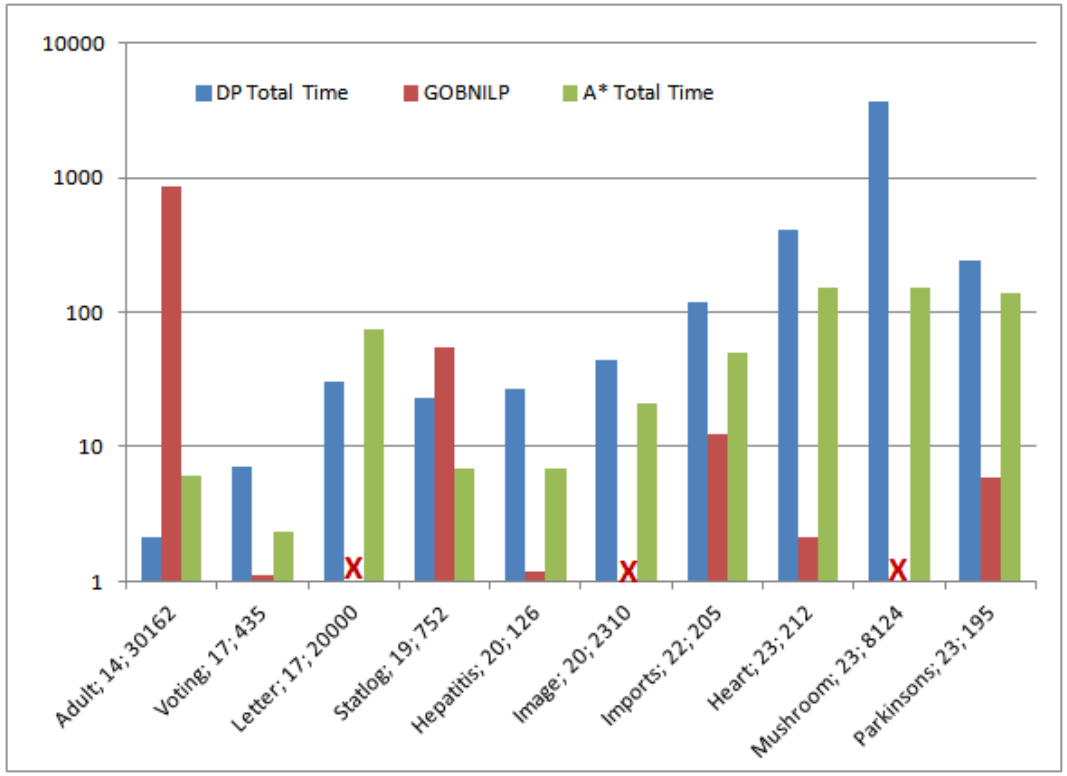
- Basics of Bayesian networks (30 minutes)
- Scoring functions (30 minutes)
- Dynamic programming (30 minutes)
- Admissible heuristic search (45 minutes)
- Integer linear programming (60 minutes)
- ➢ Empirical evaluations (15 minutes)

Number of optimal parent sets



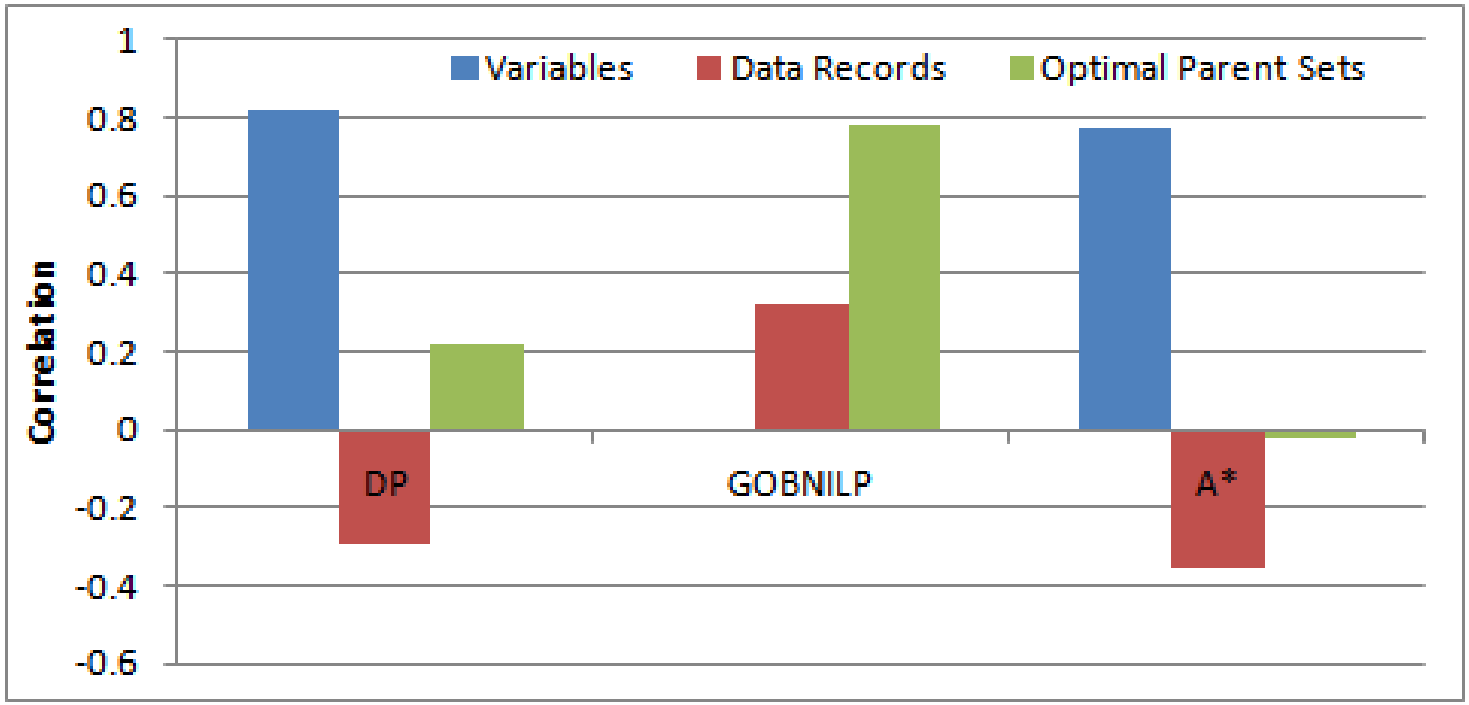
The number of parent sets and their scores stored in the full parent graphs (“Full”), the largest layer of the parent graphs in memory-efficient dynamic programming (“Largest Layer”), and the sparse representation (“Sparse”).

Comparing DP, GOBNILP, and simple A*



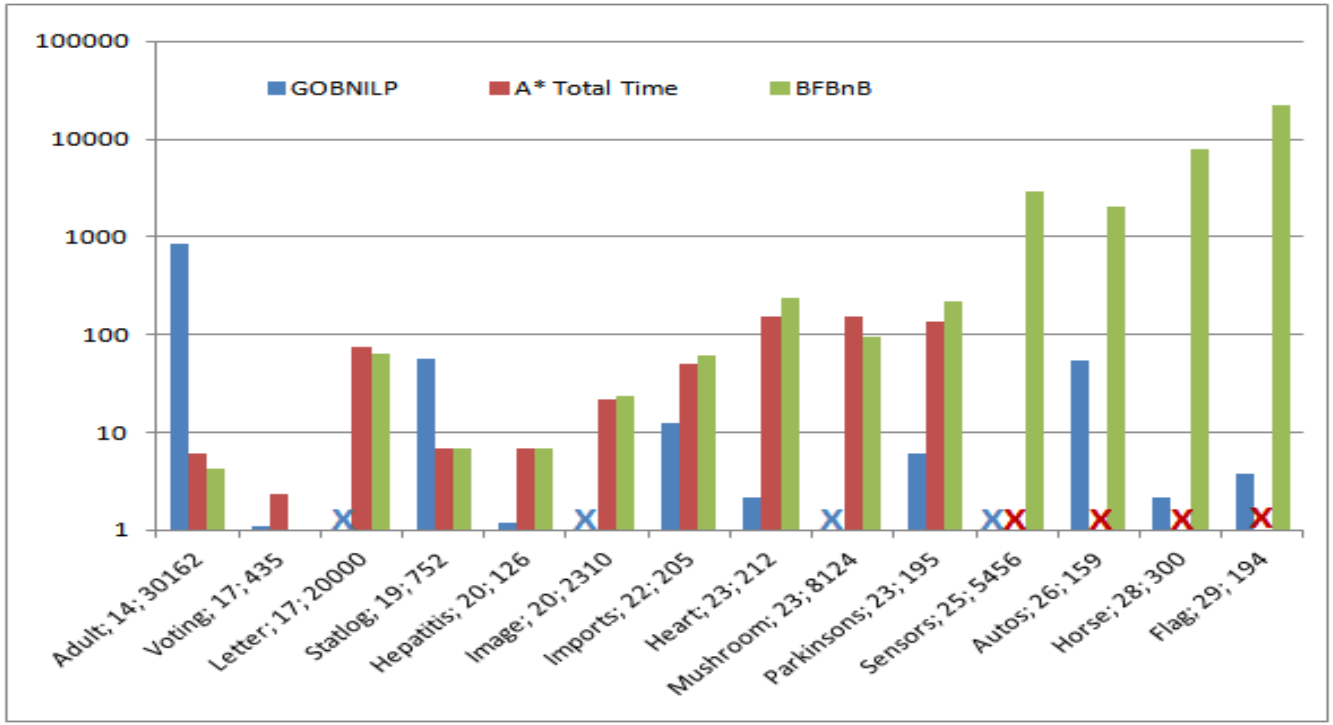
A comparison of the total time (in seconds) for BB, DP, GOBNILP, and A*. An “X” means that the corresponding algorithm did not finish within the time limit (7,200 seconds) or ran out of memory in the case of A*.

Factors affecting learning difficulty



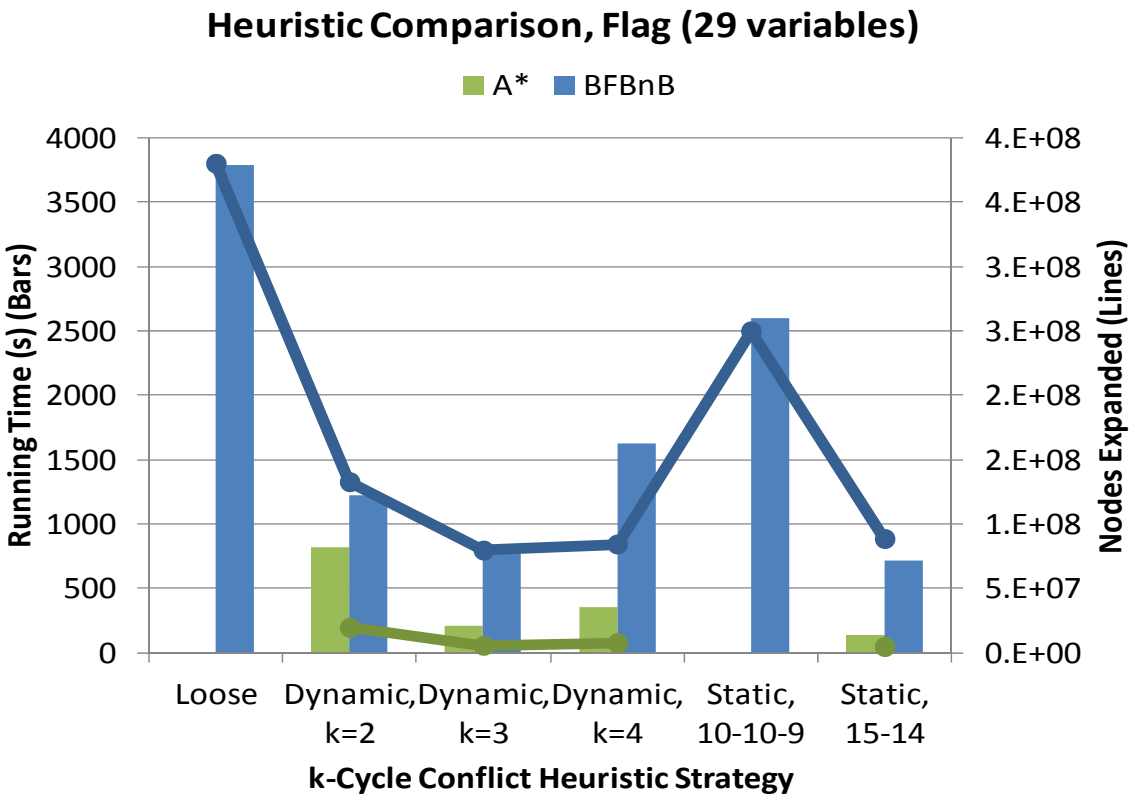
The correlation between the search time of the algorithms and several factors that may affect the difficulty of a learning problem, including the number of variables, the number of data records in a dataset, and the number of optimal parent sets.

Performance of BFBnB



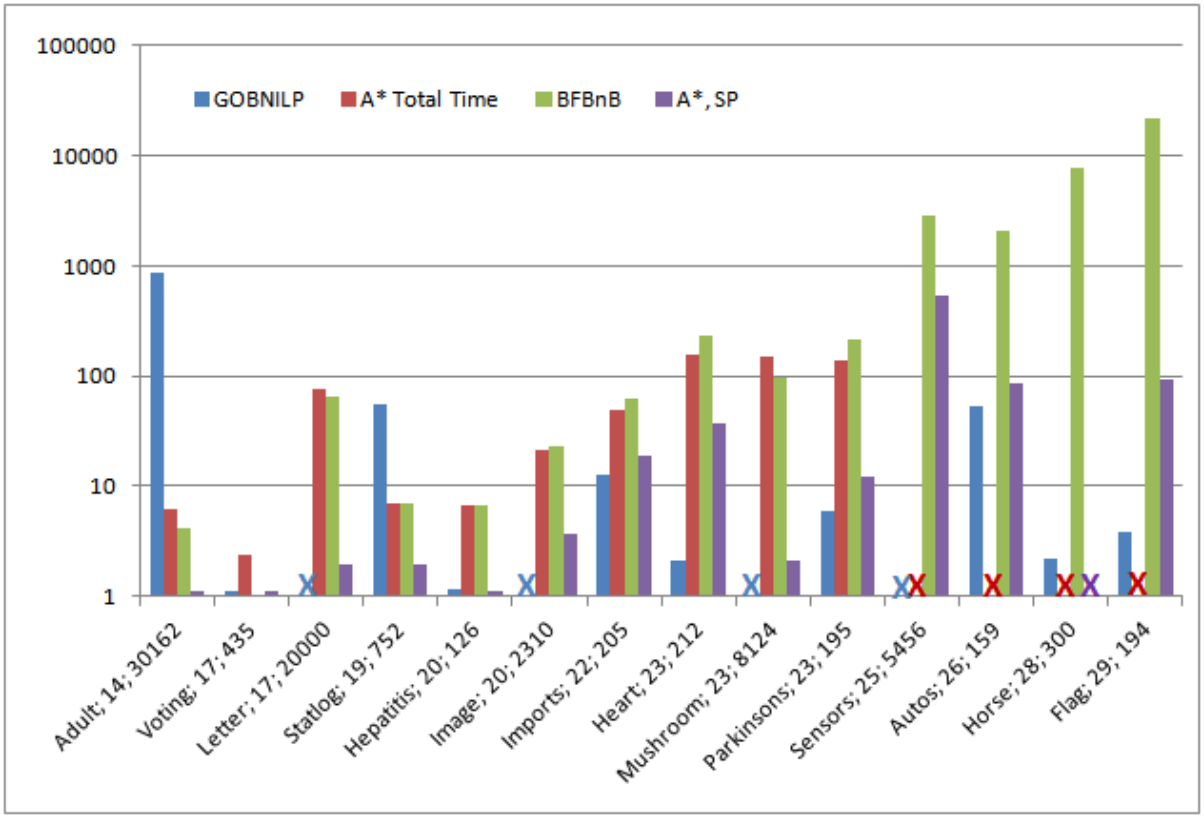
A comparison of the total time (in seconds) for GOBNILP, A*, and BFBnB. An “X” means that the corresponding algorithm did not finish within the time limit (7,200 seconds) or ran out of memory in the case of A*.

Dynamic vs static k-cycle conflict heuristic



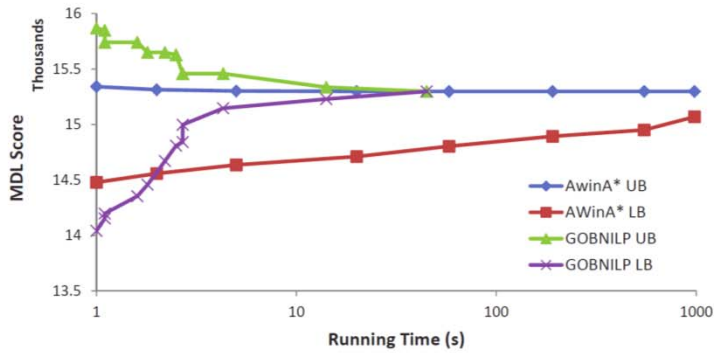
A comparison of A* enhanced with different heuristics (simple heuristic, dynamic pattern database with k = 2, 3, and 4, and static pattern databases with groupings 10-10-9 and 15-14 for the Flag dataset).

Enhancing A* with k-cycle conflict heuristic

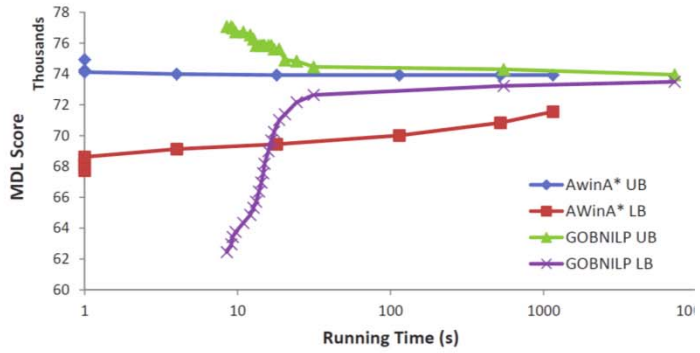


A comparison of the search time (in seconds) for GOBNILP, A*, BFBnB, and A* with pattern database heuristic. An “X” means that the corresponding algorithm did not finish within the time limit (7,200 seconds) or ran out of memory in the case of A*.

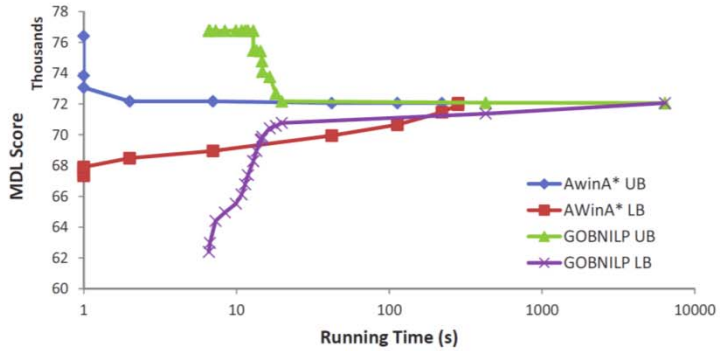
Anytime behavior comparison



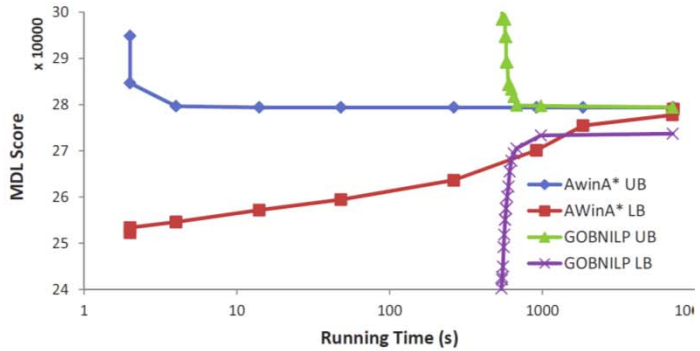
$(a) Dataset\ 29.3.1k$



$(b) Dataset\ 29.3.5k$



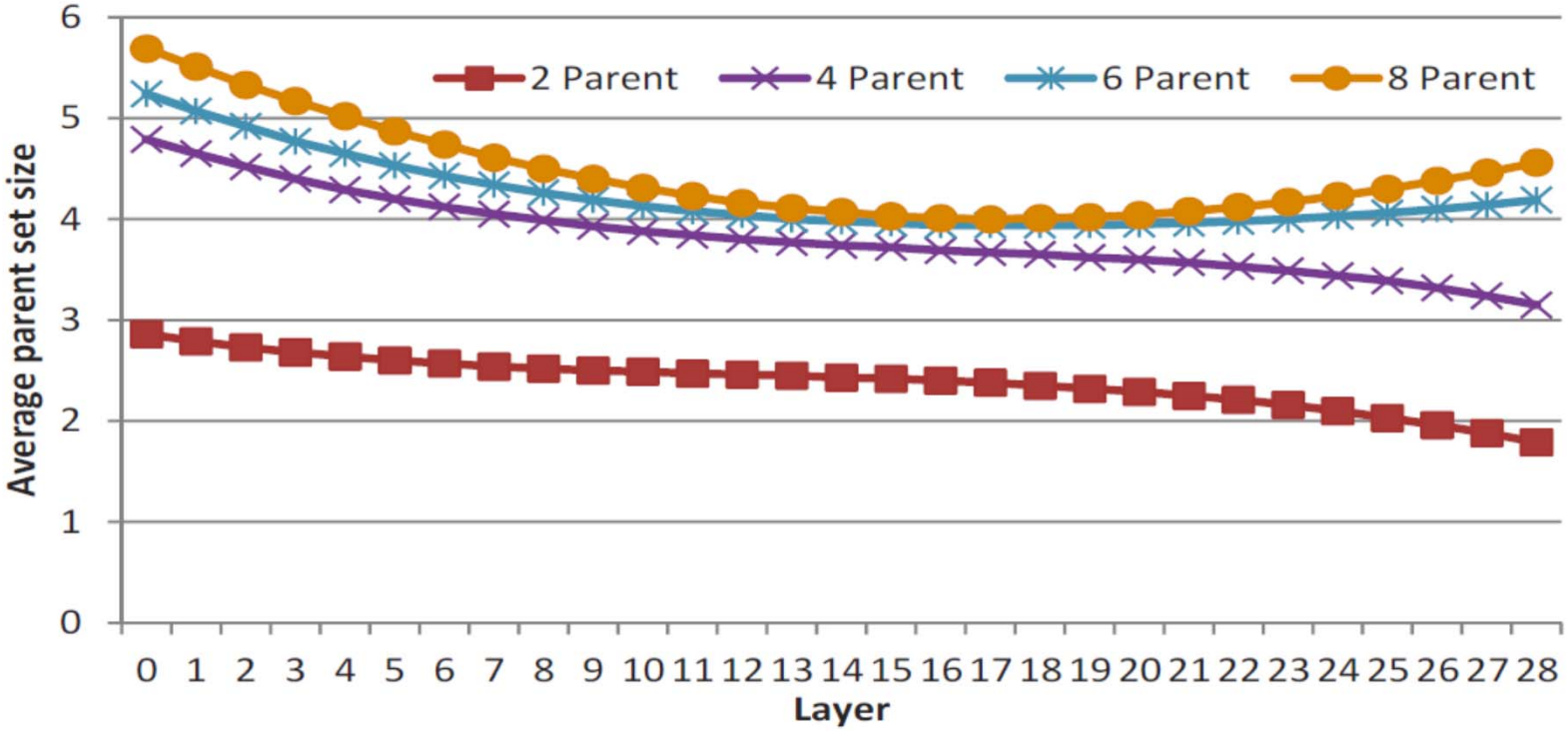
$(c) Dataset\ 29.6.5k$



$(b) Dataset\ 29.6.20k$

A comparison of the convergence of upper bounds (UB) and lower bounds (LB) for Anytime Window A* and ILP on random networks with 29 variables and {3, 6} maximum parents per variable. Then, from each network, we generated datasets with {1k, 5k, 10k, 20k} data points. We put a 2-hour (7200 seconds) time limit on all the algorithms.

Average parent size in search graph



Average parent set size of all the nodes in each layer of the order graph for the “29.*.5k” datasets.

GOBNILP 1.4

Name	p	N	Lim	Vars	Nodes	Time	To Best
insurance	27	100	3	277	1	1	1
insurance	27	1000	3	773	1	2	2
insurance	27	10000	3	3652	3	8	8
Water	32	100	3	479	1	7	7
Water	32	1000	3	573	1	6	6
Water	32	10000	3	961	1	7	7
Mildew	35	100	3	3506	1	1	1
Mildew	35	1000	3	148	1	1	1
Mildew	35	10000	3	456	1	1	1
alarm	37	100	3	906	1	1	1
alarm	37	1000	3	1927	1	4	4
alarm	37	10000	3	6472	1625	71	25

GOBNILP 1.4

Name	p	N	Lim	Vars	Nodes	Time	To Best
hailfinder	56	100	3	217	1	1	1
hailfinder	56	1000	3	758	1	8	8
hailfinder	56	10000	3	3768	17	33	33
carpo	60	100	3	5068	343	268	234
carpo	60	1000	3	3826	8	75	75
carpo	60	10000	3	16390	2327	354	198
Diabetes	413	100	2	4384	1	104	104
Diabetes	413	1000	2	21493	1	7446	7446
Pigs	441	100	2	2612	1	4	4
Pigs	441	1000	2	15847	1	33	33

Details

- All runs done using a single core of 2.5 GHz machine.
- Times in the preceding tables do not include the time taken to produce the local BDeu scores—this took 2082s for 'Pigs' with 1000 datapoints.
- Further benchmarks available at the GOBNILP home page: `http://www.cs.york.ac.uk/aig/sw/gobnilp/`

What works (eventually) and what doesn't

- For ≈ 12 BN variables, exact learning is easy and quick, and there is no node to restrict the size of parent sets.
- For the 724 node 'Link' BN we produced millions of local scores . . .
- . . . but GOBNILP ran out of memory when we tried to learn from them.
- With a parent set limit of 2, we have learned BNs with 1614 nodes

Summary

- **A tutorial on optimal algorithms for learning Bayesian network structures**
- **Topics covered:**
  - **Basics of Bayesian networks (30 minutes)**
  - **Scoring functions (30 minutes)**
  - **Dynamic programming (30 minutes)**
  - **Admissible heuristic search (45 minutes)**
  - **Integer linear programming (60 minutes)**
  - **Evaluation (15 minutes)**
- **Software packages available from**
  - **URLearning Java software:** http://url.cs.qc.cuny.edu//software/URLearning.html
  - **URLearning Weka package:** http://url.cs.qc.cuny.edu//software/URLearning.html
  - **GOBNILP:** http://www.cs.york.ac.uk/aig/sw/gobnilp/

Acknowledgements

- **Cussens**
  - **UK Medical Research Council (Project Grant G1002312)**
- **Malone**
  - **Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170)**
- **Yuan**
  - **NSF CAREER grant, IIS-0953723**
  - **NSF IIS grant, IIS-1219114**

References

- Silander, T., & Myllymaki, P. (2006). A simple approach for finding the globally optimal Bayesian network structure. In Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06), pp. 445–452. AUAI Press.
- Jaakkola, T., Sontag, D., Globerson, A., & Meila, M. (2010). Learning Bayesian network structure using LP relaxations. In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 358–365, Chia Laguna Resort,Sardinia, Italy.
- Cussens, J. (2011). Bayesian network learning with cutting planes. In Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11), pp. 153–160, Corvallis, Oregon. AUAI Press.
- Yuan, C., Malone, B., & Wu, X. (2011). Learning optimal Bayesian networks using A* search. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11), pp. 2186–2191, Helsinki, Finland.
- Malone, B., Yuan, C., Hansen, E., & Bridges, S. (2011a). Improving the scalability of optimal Bayesian network learning with frontier breadth-first branch and bound search. In Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11), pp. 479–488, Barcelona, Catalonia, Spain.
- Malone, B., Yuan, C., & Hansen, E. A. (2011b). Memory-efficient dynamic programming for learning optimal Bayesian networks. In Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11), pp. 1057–1062, San Francisco, CA.
- Yuan, C., & Malone, B. (2012). An improved admissible heuristic for learning optimal Bayesian networks. In Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12), pp. 924–933, Catalina Island, CA.
- Malone, B., & Yuan, C. (2013). Evaluating anytime algorithms for learning optimal Bayesian networks. In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13), pp. 381–390, Seattle, Washington.