Computational Complexity of Bayesian Networks and Markov Random Fields

Johan Kwisthout and Cassio P. de Campos

Radboud University Nijmegen / Queen's University Belfast

PGM'16 / OR Days'16

Probabilistic Graphical Models

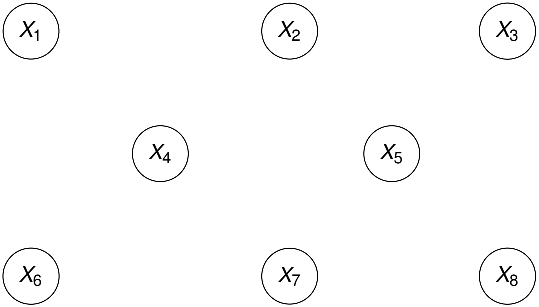
aka Decomposable Multivariate Probabilistic Models

(whose decomposability is induced by independence)

Probabilistic Graphical Models

aka Decomposable Multivariate Probabilistic Models

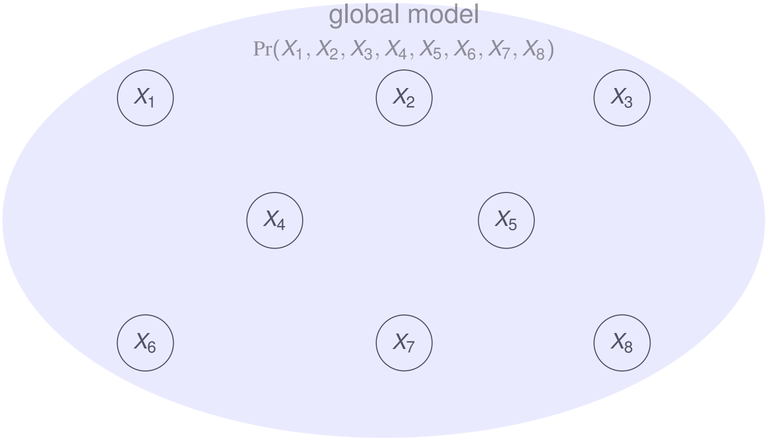
(whose decomposability is induced by independence)



Probabilistic Graphical Models

aka Decomposable Multivariate Probabilistic Models

(whose decomposability is induced by independence)

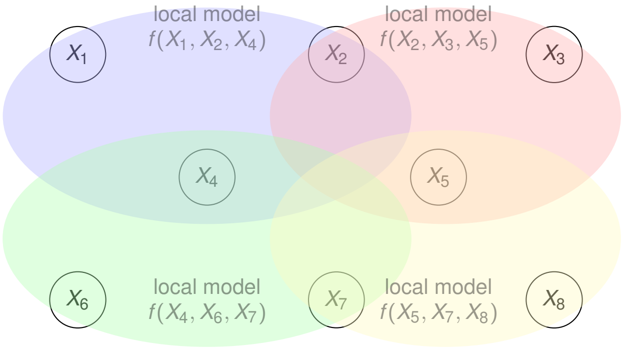


Probabilistic Graphical Models

aka Decomposable Multivariate Probabilistic Models

(whose decomposability is induced by independence)

$$\Pr(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8) \propto f(X_1, X_2, X_4) \times f(X_2, X_3, X_5) \times f(X_4, X_6, X_7) \times f(X_5, X_7, X_8)$$



Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\mathrm{Pr}(X_i \mid \mathrm{pa}(X_i)) \, , \forall i = 1, \ldots, n \, , \forall \mathrm{pa}(X_i) \}$
- Defines a joint probability mass function
  - $\mathrm{Pr}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \mathrm{Pr}(x_i \mid \mathrm{pa}(X_i))$

Bayesian networks

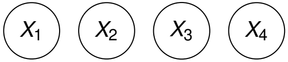
- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)), \forall i = 1, \ldots, n, \forall \mathrm{pa}(X_i)\}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$

Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

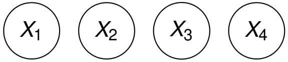
    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)), \forall i = 1, \ldots, n, \forall \mathrm{pa}(X_i)\}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

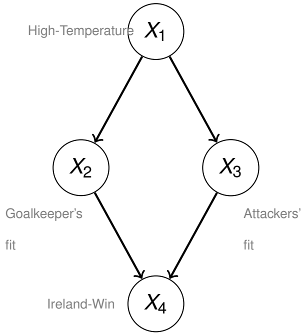
    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)), \forall i = 1, \ldots, n, \forall \mathrm{pa}(X_i)\}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

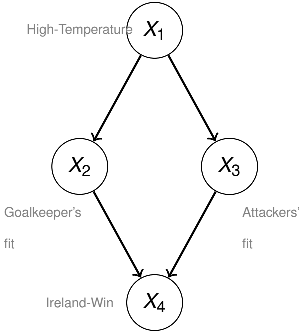
    “any node is conditionally independent of its non-descendents given its parents”
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)), \forall i = 1, \ldots, n, \forall \mathrm{pa}(X_i)\}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

    “any node is conditionally independent of its non-descendents given its parents”
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)) \,, \forall i = 1, \ldots, n \,, \forall \mathrm{pa}(X_i) \}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$

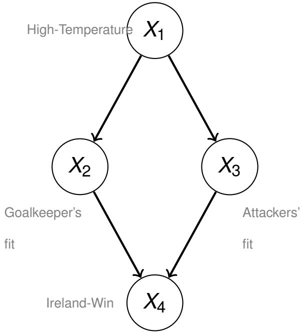


E.g., given temperature, fitnesses independent

Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

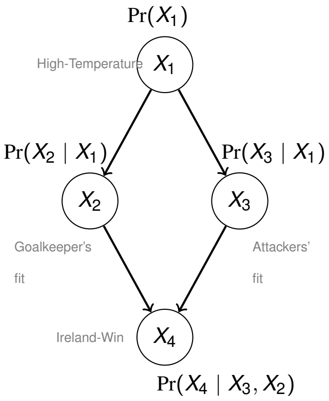
    “any node is conditionally independent of its non-descendents given its parents”
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)) \,, \forall i = 1, \ldots, n \,, \forall \mathrm{pa}(X_i) \}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

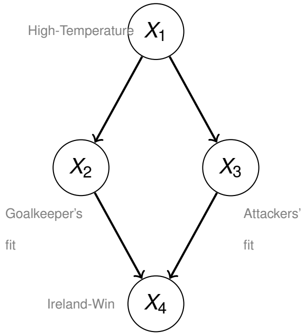
    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i))\ , \forall i = 1, \ldots, n\ , \forall \mathrm{pa}(X_i)\ \}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



Bayesian networks

- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

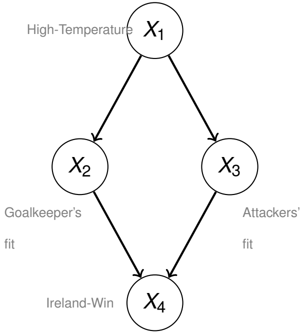
    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)) \, , \forall i = 1, \ldots, n \, , \forall \mathrm{pa}(X_i) \, \}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



Bayesian networks

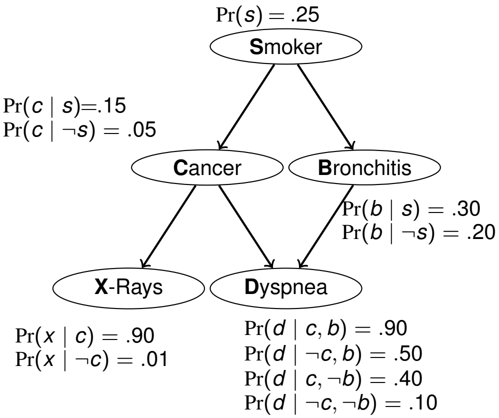
- Set of categorical variables $\mathbf{X} = X_1, \ldots, X_n$
- Directed acyclic graph
  - conditional (stochastic) independencies according to the Markov condition:

    "any node is conditionally independent of its non-descendents given its parents"
- A conditional mass function for each node and each possible value of the parents
  - $\{\Pr(X_i \mid \mathrm{pa}(X_i)) \ , \forall i = 1, \ldots, n \ , \forall \mathrm{pa}(X_i) \ \}$
- Defines a joint probability mass function
  - $\Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$



$$\Pr(x_1, x_2, x_3, x_4) =$$
$$\Pr(x_1)\Pr(x_2 \mid x_1)\Pr(x_3 \mid x_1)\Pr(x_4 \mid x_3, x_2)$$

Bayesian network: A Joint Probability Distribution



$\Pr(s) = .25$
Smoker
$\Pr(c \mid s) = .15$
$\Pr(c \mid \neg s) = .05$
Cancer
Bronchitis
$\Pr(b \mid s) = .30$
$\Pr(b \mid \neg s) = .20$
X-Rays
Dyspnea
$\Pr(x \mid c) = .90$
$\Pr(x \mid \neg c) = .01$
$\Pr(d \mid c, b) = .90$
$\Pr(d \mid \neg c, b) = .50$
$\Pr(d \mid c, \neg b) = .40$
$\Pr(d \mid \neg c, \neg b) = .10$

$$\Pr(s, c, b, x, d) = \Pr(s)\Pr(c \mid s)\Pr(b \mid s)\Pr(x \mid c)\Pr(d \mid c, b)$$

$$\Pr(s, c, b, x, d) = .25 \times .15 \times .30 \times .90 \times .90$$

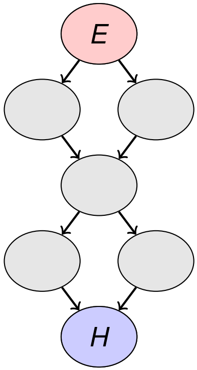
Inferences with Bayesian networks

- Conditional probs (or mode/MAP) for variables of interest $\mathbf{H}$ given observations $\mathbf{E} = \mathbf{e}$
- Inferences with Bayesian nets:
  Computing probabilities:

$$\Pr(\mathbf{h} \mid \mathbf{e}) = \frac{\Pr(\mathbf{h}, \mathbf{e})}{\Pr(\mathbf{e})} = \frac{\sum_{\mathbf{x} \setminus \{\mathbf{h},\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}{\sum_{\mathbf{x} \setminus \{\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}$$

  Finding most probable explanation ($\mathbf{H} \cup \mathbf{E} = \mathbf{X}$):

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$



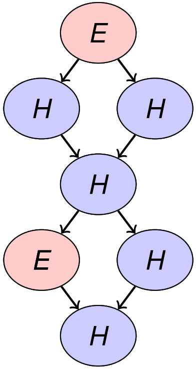
Inferences with Bayesian networks

- Conditional probs (or mode/MAP) for variables of interest $\mathbf{H}$ given observations $\mathbf{E} = \mathbf{e}$
- Inferences with Bayesian nets:
  Computing probabilities:

$$\Pr(\mathbf{h} \mid \mathbf{e}) = \frac{\Pr(\mathbf{h}, \mathbf{e})}{\Pr(\mathbf{e})} = \frac{\sum_{\mathbf{x} \setminus \{\mathbf{h}, \mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}{\sum_{\mathbf{x} \setminus \{\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}$$

  Finding most probable explanation ($\mathbf{H} \cup \mathbf{E} = \mathbf{X}$):

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$



Complexity theory

- Many computations on Bayesian networks are NP-hard
- Meaning that we shall not hope for poly time algorithms that solve **all** instances
- A better understanding of complexity allows us to
  - Get insight in what makes particular instances hard
  - Understand why and when computations can be tractable
  - Use this knowledge in practical applications
- Why go beyond NP-hardness to find exact complexity classes etc.?
  - For exactly the reasons above!
- See lecture notes for detailed background at the PGM website (available soon).

Complexity theory

- Many computations on Bayesian networks are NP-hard
- Meaning that we shall not hope for poly time algorithms that solve **all** instances
- A better understanding of complexity allows us to
    - Get insight in what makes particular instances hard
    - Understand why and when computations can be tractable
    - Use this knowledge in practical applications
- Why go beyond NP-hardness to find exact complexity classes etc.?
    - For exactly the reasons above!
- See lecture notes for detailed background at the PGM website (available soon).

Complexity theory

- Many computations on Bayesian networks are NP-hard
- Meaning that we shall not hope for poly time algorithms that solve **all** instances
- A better understanding of complexity allows us to
  - Get insight in what makes particular instances hard
  - Understand why and when computations can be tractable
  - Use this knowledge in practical applications
- Why go beyond NP-hardness to find exact complexity classes etc.?
  - For exactly the reasons above!
- See lecture notes for detailed background at the PGM website (available soon).

Complexity theory

- Many computations on Bayesian networks are NP-hard
- Meaning that we shall not hope for poly time algorithms that solve **all** instances
- A better understanding of complexity allows us to
  - Get insight in what makes particular instances hard
  - Understand why and when computations can be tractable
  - Use this knowledge in practical applications
- Why go beyond NP-hardness to find exact complexity classes etc.?
  - For exactly the reasons above!
- See lecture notes for detailed background at the PGM website (available soon).

Today's menu

- We assume you know **something** about complexity theory
  - Classes P, NP; NP-hardness
  - polynomial-time reductions
- We will build on that by adding the following concepts
  - Non-deterministic Turing machines
  - Probabilistic Turing Machines
  - Complexity class PP and PP with oracles
- We will demonstrate complexity results of
  - Inference problem (compute $\Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$)
  - MPE problem (compute $\arg\max_{\mathbf{h}} \Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$)

Today's menu

- We assume you know **something** about complexity theory
  - Classes P, NP; NP-hardness
  - polynomial-time reductions
- We will build on that by adding the following concepts
  - Non-deterministic Turing machines
  - Probabilistic Turing Machines
  - Complexity class PP and PP with oracles
- We will demonstrate complexity results of
  - Inference problem (compute $\Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$)
  - MPE problem (compute $\arg\max_{\mathbf{h}} \Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$)

Today’s menu

- We assume you know **something** about complexity theory
  - Classes P, NP; NP-hardness
  - polynomial-time reductions
- We will build on that by adding the following concepts
  - Non-deterministic Turing machines
  - Probabilistic Turing Machines
  - Complexity class PP and PP with oracles
- We will demonstrate complexity results of
  - Inference problem (compute $\Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$)
  - MPE problem (compute $\arg\max_{\mathbf{h}} \Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$)

Notation

- We use the following **notational conventions**
  - Network: $\mathcal{B} = (\mathbf{G}_{\mathcal{B}}, \Pr)$
  - Variable: X, Sets of variables: $\mathbf{X}$
  - Value assignment: x, Joint value assignment: $\mathbf{x}$
  - Evidence (observations): $\mathbf{E} = \mathbf{e}$
- Our canonical problems are SAT variants
  - Boolean formula ϕ with variables $X_1, \ldots, X_n$, possibly partitioned into subsets
  - In this context: quantifiers $\exists$ and MAJ
  - Simplest version: given ϕ, does there *exists* ($\exists$) a truth assignment to the variables that satisfies ϕ?
  - Other example: given ϕ, does the *majority* (MAJ) of truth assignments to the variables satisfy ϕ?

Notation

- We use the following **notational conventions**
  - Network: $\mathcal{B} = (\mathbf{G}_{\mathcal{B}}, \Pr)$
  - Variable: X, Sets of variables: $\mathbf{X}$
  - Value assignment: x, Joint value assignment: $\mathbf{x}$
  - Evidence (observations): $\mathbf{E} = \mathbf{e}$
- Our canonical problems are SAT variants
  - Boolean formula ϕ with variables $X_1, \ldots, X_n$, possibly partitioned into subsets
  - In this context: quantifiers $\exists$ and MAJ
  - Simplest version: given ϕ, does there *exists* ($\exists$) a truth assignment to the variables that satisfies ϕ?
  - Other example: given ϕ, does the *majority* (MAJ) of truth assignments to the variables satisfy ϕ?

Hard and Complete

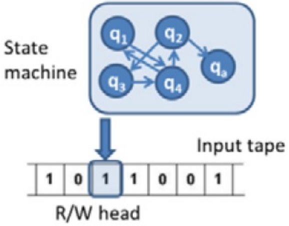
- A problem Π is **hard** for a complexity class C if every problem in C can be reduced to Π
- Reductions are polynomial-time many-one reductions
- Π is polynomial-time many-one reducible to Π' if there exists a polynomial-time computable function f such that $x \in \Pi \Leftrightarrow f(x) \in \Pi'$
- A problem Π is **complete** for a class C if it is both in C and hard for C.
- Such a problem may be regarded as being ‘at least as hard’ as any other problem in C: since we can reduce any problem in C to Π in polynomial time, a polynomial time algorithm for Π would imply a polynomial time algorithm for **every** problem in C

Hard and Complete

- A problem Π is **hard** for a complexity class C if every problem in C can be reduced to Π
- Reductions are polynomial-time many-one reductions
- Π is polynomial-time many-one reducible to Π′ if there exists a polynomial-time computable function f such that $x \in \Pi \Leftrightarrow f(x) \in \Pi'$
- A problem Π is **complete** for a class C if it is both in C and hard for C.
- Such a problem may be regarded as being ‘at least as hard’ as any other problem in C: since we can reduce any problem in C to Π in polynomial time, a polynomial time algorithm for Π would imply a polynomial time algorithm for **every** problem in C

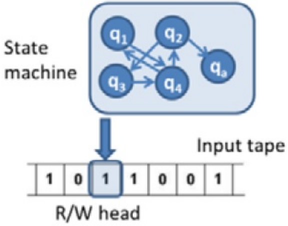
Turing machines

When computer scientists study formal aspects of computation they often use a abstracted computation device called a Turing machine that can decide decision problems. This ‘machine’ is not a realistic model of a CPU, although it could in principle simulate any CPU as its computational possibilities are **universal**: every computation can in principle be encoded and performed by a Turing Machine.



Turing machines

When computer scientists study formal aspects of computation they often use a abstracted computation device called a Turing machine that can decide decision problems. This ‘machine’ is not a realistic model of a CPU, although it could in principle simulate any CPU as its computational possibilities are **universal**: every computation can in principle be encoded and performed by a Turing Machine.



Turing machines

A Turing machine (hereafter TM), denoted by $\mathcal{M}$, consists of a one-dimensional **tape**, a read/write **head**, and a **state machine**. We have a finite set of states Q with designated accepting and rejecting states q_Y and q_N.

The **symbols** that may occur on the tape are $\Gamma = \{0, 1, b\}$,we assume that initially the tape only contains symbols $\Sigma = \{0, 1\}$.

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a **transition function** (L and R mean 'shifting tape one position to the left/right')

Formally, TMs decide *languages*, i.e., strings of ones and zeros that encode particular instances of a problem P, such as a satisfiability instance. We will in the remainder assume that such an encoding always exists and we will use problems, rather than languages.

Turing machines

A Turing machine (hereafter TM), denoted by $\mathcal{M}$, consists of a one-dimensional **tape**, a read/write **head**, and a **state machine**. We have a finite set of states Q with designated accepting and rejecting states q_Y and q_N.

The **symbols** that may occur on the tape are $\Gamma = \{0, 1, b\}$,we assume that initially the tape only contains symbols $\Sigma = \{0, 1\}$.

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a **transition function** (L and R mean 'shifting tape one position to the left/right')

Formally, TMs decide *languages*, i.e., strings of ones and zeros that encode particular instances of a problem P, such as a satisfiability instance. We will in the remainder assume that such an encoding always exists and we will use problems, rather than languages.

Turing machines

A Turing machine (hereafter TM), denoted by $\mathcal{M}$, consists of a one-dimensional **tape**, a read/write **head**, and a **state machine**. We have a finite set of states Q with designated accepting and rejecting states q_Y and q_N.

The **symbols** that may occur on the tape are $\Gamma = \{0, 1, b\}$,we assume that initially the tape only contains symbols $\Sigma = \{0, 1\}$.

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a **transition function** (L and R mean ‘shifting tape one position to the left/right’)

Formally, TMs decide *languages*, i.e., strings of ones and zeros that encode particular instances of a problem P, such as a satisfiability instance. We will in the remainder assume that such an encoding always exists and we will use problems, rather than languages.

Turing machines

A Turing machine (hereafter TM), denoted by $\mathcal{M}$, consists of a one-dimensional **tape**, a read/write **head**, and a **state machine**. We have a finite set of states Q with designated accepting and rejecting states q_Y and q_N.

The **symbols** that may occur on the tape are $\Gamma = \{0, 1, b\}$,we assume that initially the tape only contains symbols $\Sigma = \{0, 1\}$.

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a **transition function** (L and R mean ‘shifting tape one position to the left/right’)

Formally, TMs decide *languages*, i.e., strings of ones and zeros that encode particular instances of a problem P, such as a satisfiability instance. We will in the remainder assume that such an encoding always exists and we will use problems, rather than languages.

Decisions

A TM $\mathcal{M}$ **decides** a problem P if and only if, when presented with an input string x on its tape, it halts in the accepting state q_Y if $x \in P$ and it halts in the rejecting state q_N if $x \notin P$.

If the transition function δ maps every tuple (q_i, γ_k) to at most one tuple (q_j, γ_l, p), then $\mathcal{M}$ ($\mathcal{T}$) is called a **deterministic** Turing machine, else it is termed as a **non-deterministic** Turing machine.

A deterministic TM **accepts** x if its computation path accepts x, that is, ends in state q_Y. A non-deterministic TM accepts x if at least one of its possible computation paths accepts x.

The **time complexity** of deciding P by $\mathcal{M}$ is defined as the maximum number of steps that $\mathcal{M}$ uses, as a function of the size of the input x.

Decisions

A TM $\mathcal{M}$ **decides** a problem P if and only if, when presented with an input string x on its tape, it halts in the accepting state q_Y if $x \in P$ and it halts in the rejecting state q_N if $x \notin P$.

If the transition function δ maps every tuple (q_i, γ_k) to at most one tuple (q_j, γ_l, p), then $\mathcal{M}$ $(\mathcal{T})$ is called a **deterministic** Turing machine, else it is termed as a **non-deterministic** Turing machine.

A deterministic TM **accepts** x if its computation path accepts x, that is, ends in state q_Y. A non-deterministic TM accepts x if at least one of its possible computation paths accepts x.

The **time complexity** of deciding P by $\mathcal{M}$ is defined as the maximum number of steps that $\mathcal{M}$ uses, as a function of the size of the input x.

Decisions

A TM $\mathcal{M}$ **decides** a problem P if and only if, when presented with an input string x on its tape, it halts in the accepting state q_Y if $x \in P$ and it halts in the rejecting state q_N if $x \notin P$.

If the transition function δ maps every tuple (q_i, γ_k) to at most one tuple (q_j, γ_l, p), then $\mathcal{M}$ ($\mathcal{T}$) is called a **deterministic** Turing machine, else it is termed as a **non-deterministic** Turing machine.

A deterministic TM **accepts** x if its computation path accepts x, that is, ends in state q_Y. A non-deterministic TM accepts x if at least one of its possible computation paths accepts x.

The **time complexity** of deciding P by $\mathcal{M}$ is defined as the maximum number of steps that $\mathcal{M}$ uses, as a function of the size of the input x.

Decisions

A TM $\mathcal{M}$ **decides** a problem P if and only if, when presented with an input string x on its tape, it halts in the accepting state q_Y if $x \in P$ and it halts in the rejecting state q_N if $x \notin P$.

If the transition function δ maps every tuple (q_i, γ_k) to at most one tuple (q_j, γ_l, p), then $\mathcal{M}$ ($\mathcal{T}$) is called a **deterministic** Turing machine, else it is termed as a **non-deterministic** Turing machine.

A deterministic TM **accepts** x if its computation path accepts x, that is, ends in state q_Y. A non-deterministic TM accepts x if at least one of its possible computation paths accepts x.

The **time complexity** of deciding P by $\mathcal{M}$ is defined as the maximum number of steps that $\mathcal{M}$ uses, as a function of the size of the input x.

P and NP

- The class P is the class of all decision problems that are decidable on a deterministic TM in a time which is polynomial in the length of the input string x
- The class NP (*non-deterministic polynomial time*) is the class of all problems that are decidable on a *non*-deterministic TM in a time which is polynomial in the length of the input x
- Alternatively NP can be defined as the class of all problems that can be *verified* in polynomial time, measured in the size of the input x, on a deterministic TM: for any problem $P \in$ NP, there exists a TM $\mathcal{M}$ that, when provided with a tuple (x, c) on its input tape, can verify in polynomial time that c is a ‘proof’ of the fact that $x \in P$

P and NP

- The class P is the class of all decision problems that are decidable on a deterministic TM in a time which is polynomial in the length of the input string x
- The class NP (*non-deterministic polynomial time*) is the class of all problems that are decidable on a *non*-deterministic TM in a time which is polynomial in the length of the input x
- Alternatively NP can be defined as the class of all problems that can be *verified* in polynomial time, measured in the size of the input x, on a deterministic TM: for any problem $P \in$ NP, there exists a TM $\mathcal{M}$ that, when provided with a tuple (x, c) on its input tape, can verify in polynomial time that c is a 'proof' of the fact that $x \in P$

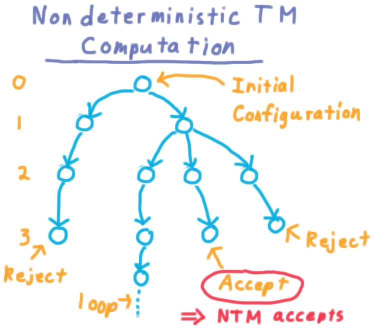
P and NP

- The class P is the class of all decision problems that are decidable on a deterministic TM in a time which is polynomial in the length of the input string x
- The class NP (*non-deterministic polynomial time*) is the class of all problems that are decidable on a *non*-deterministic TM in a time which is polynomial in the length of the input x
- Alternatively NP can be defined as the class of all problems that can be *verified* in polynomial time, measured in the size of the input x, on a deterministic TM: for any problem $P \in$ NP, there exists a TM $\mathcal{M}$ that, when provided with a tuple (x, c) on its input tape, can verify in polynomial time that c is a ‘proof’ of the fact that $x \in P$

Non-determinism

- What does it mean for an algorithm to be deterministic, respectively non-deterministic?

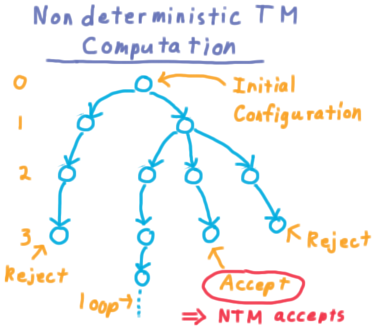
- Think of an execution of an algorithm on a particular input *i* as a *computation path*: an exact description of the state of the computation with respect to *i*



Non-determinism

- What does it mean for an algorithm to be deterministic, respectively non-deterministic?

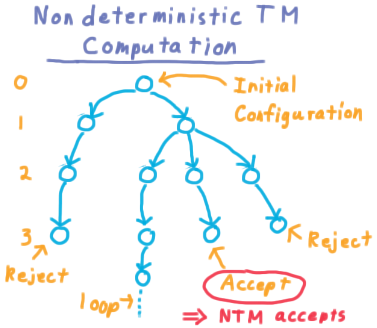
- Think of an execution of an algorithm on a particular input *i* as a *computation path*: an exact description of the state of the computation with respect to *i*



Non-determinism

- What does it mean for an algorithm to be deterministic, respectively non-deterministic?

- Think of an execution of an algorithm on a particular input *i* as a *computation path*: an exact description of the state of the computation with respect to *i*



Deterministic Find

Have a look at the following algorithms that decide whether a given integer number Z is in an array A with n elements

Algorithm 1 A deterministic algorithm that decides whether an array A contains an integer Z

```
Find(A[n], Z)
1: for $k = 1$ to $n$ do
2:     if $A[k] = Z$ then
3:         return "Yes: $A$ contains $Z$"
4:     end if
5: end for
6: return "No: $A$ does not contain $Z$"
```

Deterministic Find

Have a look at the following algorithms that decide whether a given integer number Z is in an array A with n elements

Algorithm 2 A deterministic algorithm that decides whether an array A contains an integer Z

```
Find(A[n], Z)
1: for $k = 1$ to $n$ do
2:     if $A[k] = Z$ then
3:         return "Yes: $A$ contains $Z$"
4:     end if
5: end for
6: return "No: $A$ does not contain $Z$"
```

Non-deterministic Find

Algorithm 3 A non-deterministic algorithm that decides whether an array A contains an integer Z

```
Find'(A[n], Z)
1: Guess a value of $k$ in the interval [1, ..., n];
2: if $A[k] = Z$ then
3:     return "Yes: $A$ contains $Z$"
4: end if
5: return "No: $A$ does not contain $Z$"
```

This algorithm may produce different results each time it is run on the input $\{A[n] = \{1, 2\}, Z = 2\}$, as there is a choice point in line 1, leading to *different* computation paths depending on whether the algorithm made the guess '$k = 1$' or the guess '$k = 2$'

Non-deterministic Find

For $k = 1$ the algorithm will–*incorrectly!*–return “No: A does not contain Z” and for $k = 2$ the algorithm will return “Yes: A contains Z”.

How can this algorithm be useful at all, given that it may give a wrong answer? In practice, we will not find many non-deterministic algorithms in operation, in particular because we did not impose *any* constraints on how *Find’* makes its guesses.

For all we know, *Find’* may always select $k = 1$, given the wrong output time and again! Non-deterministic algorithms are mainly theoretical constructs.

Non-deterministic Find

For $k = 1$ the algorithm will–*incorrectly!*–return “No: A does not contain Z” and for $k = 2$ the algorithm will return “Yes: A contains Z”.

How can this algorithm be useful at all, given that it may give a wrong answer? In practice, we will not find many non-deterministic algorithms in operation, in particular because we did not impose *any* constraints on how *Find’* makes its guesses.

For all we know, *Find’* may always select $k = 1$, given the wrong output time and again! Non-deterministic algorithms are mainly theoretical constructs.

Non-deterministic Find

For $k = 1$ the algorithm will–*incorrectly!*–return “No: *A* does not contain *Z*” and for $k = 2$ the algorithm will return “Yes: *A* contains *Z*”.

How can this algorithm be useful at all, given that it may give a wrong answer? In practice, we will not find many non-deterministic algorithms in operation, in particular because we did not impose *any* constraints on how *Find’* makes its guesses.

For all we know, *Find’* may always select $k = 1$, given the wrong output time and again! Non-deterministic algorithms are mainly theoretical constructs.

Non-deterministic Find

The key aspect here is the notion of *decides*. For a regular, deterministic algorithm, when there is always the same computation path on a particular input, it is clear when it correctly 'decides' an input i

For example, *Find* correctly decides $\{A[n], Z\}$ if it announces "Yes: A contains Z" if and only if A contains Z, and "No: A does not contain Z" if it does not

The notion of 'decides' in a non-deterministic algorithm is different. Here 'decides' means, that there is *at least one computation path* that accepts i if i is a *yes*-instance, and that there is *no computation path* that accepts i if i is a *no*-instance

Non-deterministic Find

The key aspect here is the notion of *decides*. For a regular, deterministic algorithm, when there is always the same computation path on a particular input, it is clear when it correctly 'decides' an input i

For example, *Find* correctly decides $\{A[n], Z\}$ if it announces "Yes: A contains Z" if and only if A contains Z, and "No: A does not contain Z" if it does not

The notion of 'decides' in a non-deterministic algorithm is different. Here 'decides' means, that there is *at least one computation path* that accepts i if i is a *yes*-instance, and that there is *no computation path* that accepts i if i is a *no*-instance

Non-deterministic Find

The key aspect here is the notion of *decides*. For a regular, deterministic algorithm, when there is always the same computation path on a particular input, it is clear when it correctly 'decides' an input i

For example, *Find* correctly decides $\{A[n], Z\}$ if it announces
"Yes: A contains Z" if and only if A contains Z, and
"No: A does not contain Z" if it does not

The notion of 'decides' in a non-deterministic algorithm is different. Here 'decides' means, that there is *at least one computation path* that accepts i if i is a *yes*-instance, and that there is *no computation path* that accepts i if i is a *no*-instance

Non-deterministic Find

In a way, a non-deterministic algorithm that accepts an input if it ends in an accepting state on at least one computation path is very powerful. Instead of testing each element of the array until it finds *Z* (what *Find* does), the non-deterministic variant *Find'* just 'branches off' and *tests every element of A in parallel*.

Think of it as follows: because each choice generates a different computation path, and only *one* computation path needs to accept, the algorithm more or less clones itself at every choice point, where each clone investigates the consequences of a particular choice.

Note, again, that non-deterministic algorithms are *theoretical constructs*, in contrast to deterministic algorithms. The algorithm does not really 'follow' any computation path; we just show that there *exists* a computation path that leads to success.

Non-deterministic Find

In a way, a non-deterministic algorithm that accepts an input if it ends in an accepting state on at least one computation path is very powerful. Instead of testing each element of the array until it finds *Z* (what *Find* does), the non-deterministic variant *Find'* just ‘branches off’ and *tests every element of A in parallel*.

Think of it as follows: because each choice generates a different computation path, and only *one* computation path needs to accept, the algorithm more or less clones itself at every choice point, where each clone investigates the consequences of a particular choice.

Note, again, that non-deterministic algorithms are *theoretical constructs*, in contrast to deterministic algorithms. The algorithm does not really ‘follow’ any computation path; we just show that there *exists* a computation path that leads to success.

Non-deterministic Find

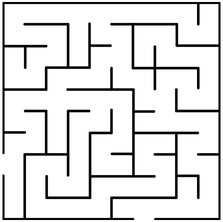
In a way, a non-deterministic algorithm that accepts an input if it ends in an accepting state on at least one computation path is very powerful. Instead of testing each element of the array until it finds *Z* (what *Find* does), the non-deterministic variant *Find'* just 'branches off' and *tests every element of A in parallel*.

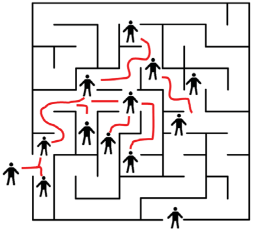
Think of it as follows: because each choice generates a different computation path, and only *one* computation path needs to accept, the algorithm more or less clones itself at every choice point, where each clone investigates the consequences of a particular choice.

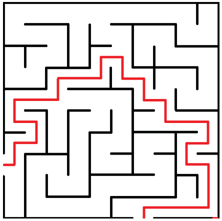
Note, again, that non-deterministic algorithms are *theoretical constructs*, in contrast to deterministic algorithms. The algorithm does not really 'follow' any computation path; we just show that there *exists* a computation path that leads to success.

Cloning or using maps

How to find your way our of this maze? By cloning yourself at every divergence, or by having a road map available?



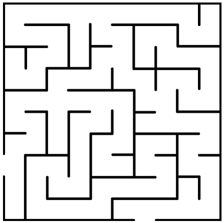
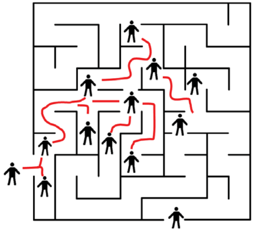
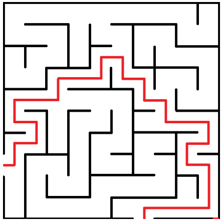




In both cases you get to the exit equally fast. This illustrates the equal definitions of NP: the set of problems that can be verified in polynomial time and the set of problems that can be decided in polynomial time with a non-deterministic algorithm

Cloning or using maps

How to find your way our of this maze? By cloning yourself at every divergence, or by having a road map available?

In both cases you get to the exit equally fast. This illustrates the equal definitions of NP: the set of problems that can be verified in polynomial time and the set of problems that can be decided in polynomial time with a non-deterministic algorithm

Deciding 3SAT

Stop and think

Recall the NP-complete 3SAT problem. What would be a suitable certificate to verify a *yes*-instance? Given this certificate, can you think of a polynomial-time, non-deterministic algorithm for deciding 3SAT?

If we think of each variable as a choice point where either TRUE or FALSE needs to be guessed, an accepting computation path will be a path that guesses a satisfying truth assignment for the formula. This is a polynomial-time, non-deterministic algorithm for deciding 3SAT: if (and only if) one of the computation paths accepts, the formula is satisfiable!

Deciding 3SAT

Stop and think

Recall the NP-complete 3SAT problem. What would be a suitable certificate to verify a *yes*-instance? Given this certificate, can you think of a polynomial-time, non-deterministic algorithm for deciding 3SAT?

If we think of each variable as a choice point where either TRUE or FALSE needs to be guessed, an accepting computation path will be a path that guesses a satisfying truth assignment for the formula. This is a polynomial-time, non-deterministic algorithm for deciding 3SAT: if (and only if) one of the computation paths accepts, the formula is satisfiable!

Probabilistic Turing Machine

- A **Probabilistic** TM (PTM) is similar to a non-deterministic TM, but the transitions are *probabilistic* rather than simply non-deterministic
- For each transition, the next state is determined stochastically according to some probability distribution
- Without loss of generality we assume that a PTM has two possible next states q_1 and q_2 at each transition, and that the next state will be q_1 with some probability p and q_2 with probability $1 - p$
- A PTM accepts a language L if the probability of ending in an accepting state, when presented an input x on its tape, is strictly larger than $^1/_2$ if and only if $x \in L$. If the transition probabilities are uniformly distributed, the machine accepts if the **majority** of its computation paths accepts
- The canonical PP-complete problem is MAJSAT: given a formula ϕ, does the majority of truth assignments satisfy it?
- It can be easily shown that NP $\subseteq$ PP.

Probabilistic Turing Machine

- A **Probabilistic** TM (PTM) is similar to a non-deterministic TM, but the transitions are *probabilistic* rather than simply non-deterministic
- For each transition, the next state is determined stochastically according to some probability distribution
- Without loss of generality we assume that a PTM has two possible next states q_1 and q_2 at each transition, and that the next state will be q_1 with some probability p and q_2 with probability $1 - p$
- A PTM accepts a language L if the probability of ending in an accepting state, when presented an input x on its tape, is strictly larger than $^1/_2$ if and only if $x \in L$. If the transition probabilities are uniformly distributed, the machine accepts if the **majority** of its computation paths accepts
- The canonical PP-complete problem is MAJSAT: given a formula ϕ, does the majority of truth assignments satisfy it?
- It can be easily shown that NP $\subseteq$ PP.

Probabilistic Turing Machine

- A **Probabilistic** TM (PTM) is similar to a non-deterministic TM, but the transitions are *probabilistic* rather than simply non-deterministic
- For each transition, the next state is determined stochastically according to some probability distribution
- Without loss of generality we assume that a PTM has two possible next states q_1 and q_2 at each transition, and that the next state will be q_1 with some probability p and q_2 with probability $1 - p$
- A PTM accepts a language L if the probability of ending in an accepting state, when presented an input x on its tape, is strictly larger than $1/2$ if and only if $x \in L$. If the transition probabilities are uniformly distributed, the machine accepts if the **majority** of its computation paths accepts
- The canonical PP-complete problem is MAJSAT: given a formula ϕ, does the majority of truth assignments satisfy it?
- It can be easily shown that NP $\subseteq$ PP.

THRESHOLD INFERENCE is PP-complete

THRESHOLD INFERENCE
Instance: A Bayesian network $\mathcal{B} = (\mathbf{G}_{\mathcal{B}}, \Pr)$, where **V** is partitioned into a set of evidence nodes **E** with a joint value assignment **e**, a set of intermediate nodes **I**, and an explanation set **H** with a joint value assignment **h**. Let $0 \leq q < 1$.
Question: Is the probability $\Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e}) > q$?

$$\Pr(\mathbf{h} \mid \mathbf{e}) = \frac{\Pr(\mathbf{h}, \mathbf{e})}{\Pr(\mathbf{e})} = \frac{\sum_{\mathbf{x} \setminus \{\mathbf{h}, \mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}{\sum_{\mathbf{x} \setminus \{\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}$$

- In this lecture we will show that THRESHOLD INFERENCE is PP-complete, meaning
  - THRESHOLD INFERENCE is in PP, and
  - THRESHOLD INFERENCE is PP-hard

THRESHOLD INFERENCE is in PP

- To show that THRESHOLD INFERENCE is in PP, we argue that THRESHOLD INFERENCE can be decided in polynomial time by a Probabilistic Turing Machine
- For brevity we will assume no evidence, i.e., the question we answer is: Given a network $\mathcal{B}$ with designated sets **H** and **H**, and $0 \leq q < 1$, is the probability $\Pr(\mathbf{H} = \mathbf{h}) > q$?
- We construct a PTM $\mathcal{M}$ such that, on such an input, it arrives in an accepting state with probability strictly larger than 1/2 if and only if $\Pr(\mathbf{h}) > q$.
- $\mathcal{M}$ computes a joint probability $\Pr(y_1, \ldots, y_n)$ by iterating over i using a topological sort of the graph, and choosing a value for each variable Y_i conform the probability distribution in its CPT given the values that are already assigned to the parents of Y_i.

THRESHOLD INFERENCE is in PP

- To show that THRESHOLD INFERENCE is in PP, we argue that THRESHOLD INFERENCE can be decided in polynomial time by a Probabilistic Turing Machine
- For brevity we will assume no evidence, i.e., the question we answer is: Given a network $\mathcal{B}$ with designated sets **H** and **H**, and $0 \leq q < 1$, is the probability $\Pr(\mathbf{H} = \mathbf{h}) > q$?
- We construct a PTM $\mathcal{M}$ such that, on such an input, it arrives in an accepting state with probability strictly larger than $1/2$ if and only if $\Pr(\mathbf{h}) > q$.
- $\mathcal{M}$ computes a joint probability $\Pr(y_1, \ldots, y_n)$ by iterating over i using a topological sort of the graph, and choosing a value for each variable Y_i conform the probability distribution in its CPT given the values that are already assigned to the parents of Y_i.

Threshold Inference is in PP

- To show that Threshold Inference is in PP, we argue that Threshold Inference can be decided in polynomial time by a Probabilistic Turing Machine
- For brevity we will assume no evidence, i.e., the question we answer is: Given a network $\mathcal{B}$ with designated sets **H** and **H**, and $0 \leq q < 1$, is the probability $\Pr(\mathbf{H} = \mathbf{h}) > q$?
- We construct a PTM $\mathcal{M}$ such that, on such an input, it arrives in an accepting state with probability strictly larger than $1/2$ if and only if $\Pr(\mathbf{h}) > q$.
- $\mathcal{M}$ computes a joint probability $\Pr(y_1, \ldots, y_n)$ by iterating over i using a topological sort of the graph, and choosing a value for each variable Y_i conform the probability distribution in its CPT given the values that are already assigned to the parents of Y_i.

THRESHOLD INFERENCE is in PP

- Each computation path then corresponds to a specific joint value assignment to the variables in the network, and the probability of arriving in a particular state corresponds with the probability of that assignment.
- After iteration, we accept with probability $1/2 + (1 - q) \cdot \epsilon$, if the joint value assignment to $Y_1, \ldots, Y_n$ is consistent with $\mathbf{h}$, and we accept with probability $1/2 - q \cdot \epsilon$ if the joint value assignment is *not* consistent with $\mathbf{h}$.
- The probability of entering an accepting state is hence $\Pr(\mathbf{h}) \cdot (1/2 + (1 - q)\epsilon) + (1 - \Pr(\mathbf{h})) \cdot (1/2 - q \cdot \epsilon) = 1/2 + \Pr(\mathbf{h}) \cdot \epsilon - q \cdot \epsilon$.
- Indeed the probability of arriving in an accepting state is strictly larger than $1/2$ if and only if $\Pr(\mathbf{h}) > q$.

THRESHOLD INFERENCE is in PP

- Each computation path then corresponds to a specific joint value assignment to the variables in the network, and the probability of arriving in a particular state corresponds with the probability of that assignment.
- After iteration, we accept with probability $1/2 + (1 - q) \cdot \epsilon$, if the joint value assignment to $Y_1, \ldots, Y_n$ is consistent with $\mathbf{h}$, and we accept with probability $1/2 - q \cdot \epsilon$ if the joint value assignment is *not* consistent with $\mathbf{h}$.
- The probability of entering an accepting state is hence $\Pr(\mathbf{h}) \cdot (1/2 + (1 - q)\epsilon) + (1 - \Pr(\mathbf{h})) \cdot (1/2 - q \cdot \epsilon) = 1/2 + \Pr(\mathbf{h}) \cdot \epsilon - q \cdot \epsilon$.
- Indeed the probability of arriving in an accepting state is strictly larger than $1/2$ if and only if $\Pr(\mathbf{h}) > q$.

Threshold Inference is in PP

- Each computation path then corresponds to a specific joint value assignment to the variables in the network, and the probability of arriving in a particular state corresponds with the probability of that assignment.
- After iteration, we accept with probability $^1\!/_2 + (1 - q) \cdot \epsilon$, if the joint value assignment to $Y_1, \ldots, Y_n$ is consistent with $\mathbf{h}$, and we accept with probability $^1\!/_2 - q \cdot \epsilon$ if the joint value assignment is *not* consistent with $\mathbf{h}$.
- The probability of entering an accepting state is hence $\Pr(\mathbf{h}) \cdot (^1\!/_2 + (1 - q)\epsilon) + (1 - \Pr(\mathbf{h})) \cdot (^1\!/_2 - q \cdot \epsilon) = {}^1\!/_2 + \Pr(\mathbf{h}) \cdot \epsilon - q \cdot \epsilon$.
- Indeed the probability of arriving in an accepting state is strictly larger than $^1\!/_2$ if and only if $\Pr(\mathbf{h}) > q$.

THRESHOLD INFERENCE is PP-hard

- We now show that THRESHOLD INFERENCE is PP-hard. We do so by reducing MAJSAT, which is known to be PP-complete, to THRESHOLD INFERENCE
- We construct a Bayesian network $\mathcal{B}_\phi$ from a given Boolean formula ϕ with n variables as follows:
  - For each propositional variable x_i in ϕ, a binary stochastic variable X_i is added to $\mathcal{B}_\phi$, with possible values TRUE and FALSE and a uniform probability distribution.
  - For each logical operator in ϕ, an additional binary variable in $\mathcal{B}_\phi$ is introduced, whose parents are the variables that correspond to the input of the operator, and whose CPT is equal to the truth table of that operator
  - The top-level operator in ϕ is denoted as V_ϕ.
- On the next slide, the network $\mathcal{B}_\phi$ is shown for the formula $\neg(x_1 \vee x_2) \vee \neg x_3$.

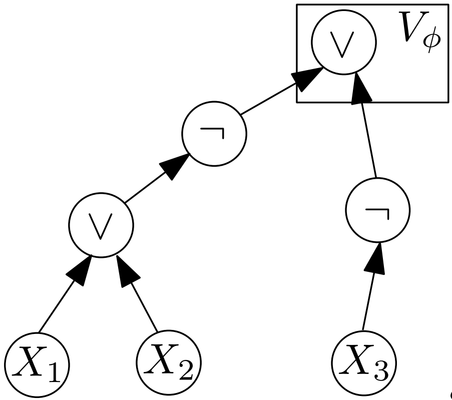
THRESHOLD INFERENCE is PP-hard

- We now show that THRESHOLD INFERENCE is PP-hard. We do so by reducing MAJSAT, which is known to be PP-complete, to THRESHOLD INFERENCE
- We construct a Bayesian network $\mathcal{B}_\phi$ from a given Boolean formula ϕ with n variables as follows:
  - For each propositional variable x_i in ϕ, a binary stochastic variable X_i is added to $\mathcal{B}_\phi$, with possible values TRUE and FALSE and a uniform probability distribution.
  - For each logical operator in ϕ, an additional binary variable in $\mathcal{B}_\phi$ is introduced, whose parents are the variables that correspond to the input of the operator, and whose CPT is equal to the truth table of that operator
  - The top-level operator in ϕ is denoted as V_ϕ.
- On the next slide, the network $\mathcal{B}_\phi$ is shown for the formula $\neg(x_1 \vee x_2) \vee \neg x_3$.

THRESHOLD INFERENCE is PP-hard

- We now show that THRESHOLD INFERENCE is PP-hard. We do so by reducing MAJSAT, which is known to be PP-complete, to THRESHOLD INFERENCE
- We construct a Bayesian network $\mathcal{B}_\phi$ from a given Boolean formula ϕ with n variables as follows:
    - For each propositional variable x_i in ϕ, a binary stochastic variable X_i is added to $\mathcal{B}_\phi$, with possible values TRUE and FALSE and a uniform probability distribution.
    - For each logical operator in ϕ, an additional binary variable in $\mathcal{B}_\phi$ is introduced, whose parents are the variables that correspond to the input of the operator, and whose CPT is equal to the truth table of that operator
    - The top-level operator in ϕ is denoted as V_ϕ.
- On the next slide, the network $\mathcal{B}_\phi$ is shown for the formula $\neg(x_1 \vee x_2) \vee \neg x_3$.

THRESHOLD INFERENCE is PP-hard



$\phi = \neg(x_1 \vee x_2) \vee \neg x_3$

THRESHOLD INFERENCE is PP-hard

- Now, for an arbitrary truth assignment **x** to the set of all propositional variables **X** in the formula ϕ we have that $\Pr(V_\phi = \text{TRUE} \mid \mathbf{X} = \mathbf{x})$ equals 1 if **x** satisfies ϕ, and 0 if **x** does not satisfy ϕ.
- Without any given joint value assignment, the prior probability $\Pr(V_\phi = \text{TRUE})$ is $\frac{\#_\phi}{2^n}$, where $\#_\phi$ is the number of satisfying truth assignments of the set of propositional variables **X**.
- Note that the above network $\mathcal{B}_\phi$ can be constructed from ϕ in polynomial time.
- We reduce MAJSAT to THRESHOLD INFERENCE. Let ϕ be a MAJSAT-instance and let $\mathcal{B}_\phi$ be the network as constructed above. Now, $\Pr(V_\phi = \text{TRUE}) > 1/2$ if and only if the majority of truth assignments satisfy ϕ.

THRESHOLD INFERENCE is PP-hard

- Now, for an arbitrary truth assignment **x** to the set of all propositional variables **X** in the formula ϕ we have that $\Pr(V_\phi = \text{TRUE} \mid \mathbf{X} = \mathbf{x})$ equals 1 if **x** satisfies ϕ, and 0 if **x** does not satisfy ϕ.
- Without any given joint value assignment, the prior probability $\Pr(V_\phi = \text{TRUE})$ is $\frac{\#_\phi}{2^n}$, where $\#_\phi$ is the number of satisfying truth assignments of the set of propositional variables **X**.
- Note that the above network $\mathcal{B}_\phi$ can be constructed from ϕ in polynomial time.
- We reduce MAJSAT to THRESHOLD INFERENCE. Let ϕ be a MAJSAT-instance and let $\mathcal{B}_\phi$ be the network as constructed above. Now, $\Pr(V_\phi = \text{TRUE}) > 1/2$ if and only if the majority of truth assignments satisfy ϕ.

THRESHOLD INFERENCE is PP-hard

- Now, for an arbitrary truth assignment **x** to the set of all propositional variables **X** in the formula ϕ we have that $\Pr(V_\phi = \text{TRUE} \mid \mathbf{X} = \mathbf{x})$ equals 1 if **x** satisfies ϕ, and 0 if **x** does not satisfy ϕ.
- Without any given joint value assignment, the prior probability $\Pr(V_\phi = \text{TRUE})$ is $\frac{\#_\phi}{2^n}$, where $\#_\phi$ is the number of satisfying truth assignments of the set of propositional variables **X**.
- Note that the above network $\mathcal{B}_\phi$ can be constructed from ϕ in polynomial time.
- We reduce MAJSAT to THRESHOLD INFERENCE. Let ϕ be a MAJSAT-instance and let $\mathcal{B}_\phi$ be the network as constructed above. Now, $\Pr(V_\phi = \text{TRUE}) > 1/2$ if and only if the majority of truth assignments satisfy ϕ.

THRESHOLD INFERENCE is PP-complete

- Given that THRESHOLD INFERENCE is PP-hard and in PP, it is PP-complete
- It is easy to show that NP $\subseteq$ PP and that THRESHOLD INFERENCE is NP-hard
- Why the additional work to prove exact complexity class?
  - PP is a class of a different nature than NP. This has effect on approximation strategies, fixed parameter tractability, etc.
  - Proving completeness for ‘higher’ complexity classes will typically also give intractability results for constrained problems.

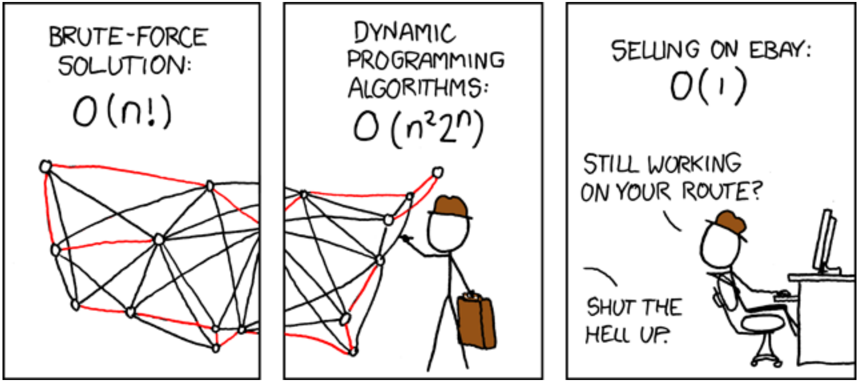
THRESHOLD INFERENCE is PP-complete

- Given that THRESHOLD INFERENCE is PP-hard and in PP, it is PP-complete
- It is easy to show that NP $\subseteq$ PP and that THRESHOLD INFERENCE is NP-hard
- Why the additional work to prove exact complexity class?
  - PP is a class of a different nature than NP. This has effect on approximation strategies, fixed parameter tractability, etc.
  - Proving completeness for 'higher' complexity classes will typically also give intractability results for constrained problems.

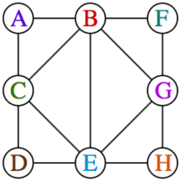
THRESHOLD INFERENCE is PP-complete

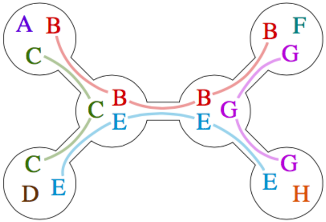
- Given that THRESHOLD INFERENCE is PP-hard and in PP, it is PP-complete
- It is easy to show that NP $\subseteq$ PP and that THRESHOLD INFERENCE is NP-hard
- Why the additional work to prove exact complexity class?
  - PP is a class of a different nature than NP. This has effect on approximation strategies, fixed parameter tractability, etc.
  - Proving completeness for ‘higher’ complexity classes will typically also give intractability results for constrained problems.

Where do lie the real-world problems



Bayesian network tree-decomposition





Source: wikipedia

Complexity of problems under some restrictions

THRESHOLD INFERENCE is:

- Bayesian network has bounded treewidth: EASY (in P)
- Bayesian network is a polytree/tree: EASY (in P)
- There is no evidence (no observed nodes): PP-complete
- Variables have bounded cardinality: PP-complete
- Nodes are binary and parameters satisfy the following condition:
  - Root nodes are associated to marginal distributions;
  - Non-root nodes are associated to Boolean operators ($\wedge$, $\vee$, $\neg$): PP-complete (even if only $\wedge$ or only $\vee$ are allowed)

MPE

THRESHOLD MPE
Instance: A Bayesian network $\mathcal{B} = (\mathbf{G}_{\mathcal{B}}, \Pr)$, where **V** is partitioned into a set of evidence nodes **E** with a joint value assignment **e** and an explanation set **H**. Let $0 \leq q < 1$.
Question: Is there **h** such that $\Pr(\mathbf{H} = \mathbf{h}, \mathbf{E} = \mathbf{e}) > q$?

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$

MPE is NP-complete

Pertinence in NP is immediate, as given **h** (the so-called certificate), we can check whether $\Pr(\mathbf{H} = \mathbf{h}, \mathbf{E} = \mathbf{e}) > q$ in polynomial time.

(Question to think about: if MPE were defined with conditional probability $\Pr(\mathbf{H} = \mathbf{h} | \mathbf{E} = \mathbf{e}) > q$, then would it still be in NP?)

Hardness: Reduction comes from an NP-hard problem: given 3-CNF propositional $\phi(X_1, \ldots, X_n)$, is there an assignment to **X** that satisfies ϕ?

MPE is NP-complete

The transformation is as follows. For each Boolean variable X_i, build a root node such that $\Pr(X_i = \text{TRUE}) = 1/2$. For each clause C_j with literals x_a, x_b, x_c (note that literals might be positive or negative), build a disjunction node Y_{abc} with parents X_a, X_b and X_c, that is, the probability function is defined such that $Y_{abc} \Leftrightarrow X_a \vee X_b \vee X_c$. Now set all non-root nodes to be observed at their true state, that is, $\mathbf{e} = \{Y_{abc} = \text{TRUE}\}_{\forall abc}$.

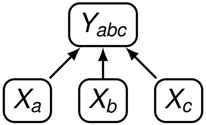


Figure: Building block representing a 3-CNF clause $(x_a \vee x_b \vee x_c)$.

Define all root nodes as **H** and ask whether there is **h** such that $\Pr(\mathbf{H} = \mathbf{h}, \mathbf{E} = \mathbf{e}) > 0$, which is true if and only if there is a satisfying assignment for the propositional formula.

Partial MPE/MAP

- Computing probabilities:

$$\Pr(\mathbf{h} \mid \mathbf{e}) = \frac{\Pr(\mathbf{h}, \mathbf{e})}{\Pr(\mathbf{e})} = \frac{\sum_{\mathbf{x} \setminus \{\mathbf{h},\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}{\sum_{\mathbf{x} \setminus \{\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}$$

- Finding most probable explanation:

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$

- Finding *partial* most probable explanation ($\mathbf{H} \cup \mathbf{E} \subset \mathbf{X}$):

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \sum_{\mathbf{x} \setminus \{\mathbf{h},\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$

- Threshold Partial MPE/MAP is $\mathrm{NP}^{\mathrm{PP}}$-complete.

Partial MPE/MAP

- Computing probabilities:

$$\Pr(\mathbf{h} \mid \mathbf{e}) = \frac{\Pr(\mathbf{h}, \mathbf{e})}{\Pr(\mathbf{e})} = \frac{\sum_{\mathbf{x} \setminus \{\mathbf{h},\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}{\sum_{\mathbf{x} \setminus \{\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))}$$

- Finding most probable explanation:

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$

- Finding *partial* most probable explanation ($\mathbf{H} \cup \mathbf{E} \subset \mathbf{X}$):

$$\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e}) = \arg\max_{\mathbf{h}} \Pr(\mathbf{h}, \mathbf{e}) = \arg\max_{\mathbf{h}} \sum_{\mathbf{x} \setminus \{\mathbf{h},\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$

- Threshold Partial MPE/MAP is $\mathrm{NP}^{\mathrm{PP}}$-complete.

Partial MPE/MAP

$$\arg\max_{\mathbf{h}} \sum_{\mathbf{x}\setminus\{\mathbf{h},\mathbf{e}\}} \prod_{i=1}^{n} \Pr(x_i \mid \mathrm{pa}(X_i))$$

The maximisation over **h** can be seen as a selector of which $\Pr(x_i \mid \mathrm{pa}(X_i))$ to use. Hence we could also write this as:

$$\arg\max_{\mathbf{v}} \sum_{\mathbf{x}} \prod_{i=1}^{n} w(x_i, \mathrm{pa}(X_i)) \cdot v(x_i, \mathrm{pa}(X_i))$$

subject to linear/integer constraints $g_k(\mathbf{v}) \leq c_k$, for $k = 1, \ldots$.
This is similar to stochastic combinatorial optimisation problems.

Markov Random Fields

- Variables $X_1, \ldots, X_n$.
- Set of positive functions $f_1, \ldots, f_m$ over subsets of the variable set.

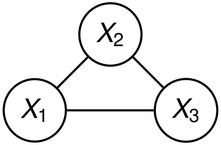


Figure: $f(X_1, X_2, X_3)$ is a positive function. We could have $f(X_1, X_2)$, $f(X_1, X_3)$, etc.

Markov Random Fields

Results in general can be translated to MRFs:

- **Hardness of problems in MRFs**: take the moralized Bayesian network as starting point of the proofs and the conditional probability functions as MRF's potentials.
- **Easiness of problems in MRFs:** build a Bayesian network creating an additional binary node for each potential (this node is the child of all nodes involved in the potential) and set the probability function for the true state of the new node as the potential of the MRF. Set evidence in these nodes to true, accordingly.

Transformation

- **BN to MRF:** Simply set $\Pr(X \mid \mathrm{pa}(X))$ as $f(X, \mathrm{pa}(X))$ and we are done.
- **MRF to BN:**

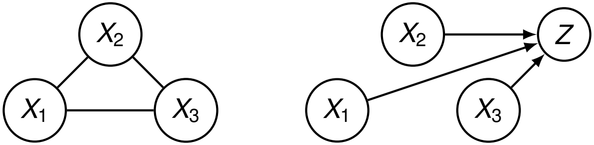
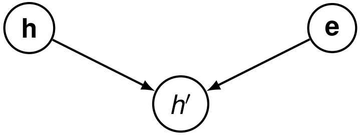


Figure: Define $\Pr(z \mid X_1, X_2, X_3) = f(X_1, X_2, X_3)$.

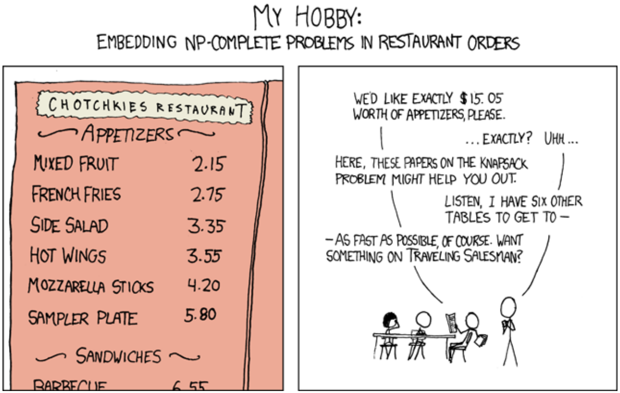
Then query $\Pr(x \mid \mathbf{e})$ of MRF becomes a query $\Pr(x \mid z, \mathbf{e})$ in the BN. Similarly, $\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e})$ of MRF is $\arg\max_{\mathbf{h}} \Pr(\mathbf{h} \mid \mathbf{e})$ in the BN.

Transformation (II)

- But we have only proved that $\Pr(h) > q$ is in PP for BNs. Now what?
- $\Pr(h \mid e) > q \iff \Pr(h, e) > q \cdot \Pr(e) \iff$ $1 \cdot \Pr(h, e) + q \cdot \Pr(h, \neg e) + q \cdot \Pr(\neg h, \neg e) > q$. So define:
  - $\Pr(h' \mid h, e) = 1$.
  - $\Pr(h' \mid \neg h, \neg e) = \Pr(h' \mid h, \neg e) = q$.
  - $\Pr(h' \mid \neg h, e) = 0$.
- Hence $\Pr(h') > q \iff \Pr(h \mid e) > q$.



Thanks


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
BARBECUE 6.55
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO –
– AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Thank you for your attention. Further questions:
j.kwisthout@donders.ru.nl, *c.decampos@qub.ac.uk*

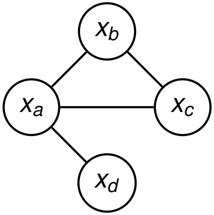
THRESHOLD INFERENCE is PP-hard in very restricted nets

THRESHOLD INFERENCE in bipartite two-layer binary Bayesian networks with no evidence and nodes defined either as marginal uniform distributions or as the disjunction $\vee$ operator is PP-hard (using only the conjunction $\wedge$ also gets there).

We reduce MAJ-2MONSAT, which is PP-complete [Roth 1996], to THRESHOLD INFERENCE:

> ***Input:*** *A 2-CNF formula $\phi(X_1, \ldots, X_n)$ with m clauses where all literals are positive.*
> ***Question:*** *Does the majority of the assignments to $X_1, \ldots, X_n$ satisfy ϕ?*

The transformation is as follows. For each Boolean variable X_i, build a root node such that $\Pr(X_i = \text{TRUE}) = 1/2$. For each clause C_j with literals x_a and x_b (note that literals are always positive), build a disjunction node Y_{ab} with parents X_a and X_b, that is, $Y_{ab} \Leftrightarrow X_a \vee X_b$. Now set all non-root nodes to be queried at their true state, that is, $\mathbf{h} = \{Y_{ab} = \text{TRUE}\}_{\forall ab}$.



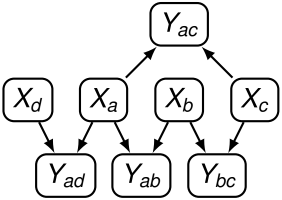
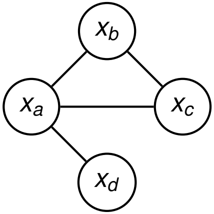
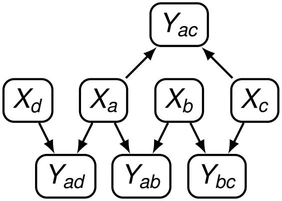


Figure: A Bayesian network (on the right) and the clauses as edges (on the left): $(x_a \vee x_b), (x_a \vee x_c), (x_a \vee x_d), (x_b \vee x_c)$.





So with this specification for **h** fixed to TRUE, at least one of the parents of each of them must be set to TRUE too. These are exactly the satisfying assignments of the propositional formula, so $\Pr(\mathbf{H} = \mathbf{h} \mid \mathbf{E} = \mathbf{e})$ for empty **E** is exactly the percentage of satisfying assignments, with $\mathbf{H} = \mathbf{Y}$ and $\mathbf{h} = \text{TRUE}$.

$\Pr(\mathbf{H} = \mathbf{h}) = \sum_{\mathbf{x}} \Pr(\mathbf{Y} = \text{TRUE} \mid \mathbf{x}) \Pr(\mathbf{x}) = \frac{1}{2^n} \sum_{\mathbf{x}} \Pr(\mathbf{Y} = \text{TRUE} \mid \mathbf{x}) > 1/2$ if and only if the majority of the assignments satisfy the formula.