Uncertain Data Management
Extensional And Intensional Query Processing

Antoine Amarilli[1], **Silviu Maniu**[2]

[1]Télécom ParisTech

[2]Université Paris-Sud

January 9th, 2017

Independent Queries

When are two queries independent?

Independent Queries

When are two queries independent?

- take two relational atoms L_1 and L_2

Independent Queries

When are two queries independent?

- take two relational atoms L_1 and L_2
- they *unify* if we find substitutions under which the two atoms become the same, i.e., they have a common image

Independent Queries

When are two queries independent?

- take two relational atoms L_1 and L_2
- they *unify* if we find substitutions under which the two atoms become the same, i.e., they have a common image
- two queries Q_1 and Q_2 are independent if no two atoms unify

Independent Queries

$Q_1 = R(x,a), S(x,b)$ and $Q_2 = R(b,a), S(c,y)$ independent?

Independent Queries

$Q_1 = R(x, a), S(x, b)$ and $Q_2 = R(b, a), S(c, y)$ independent?

- both have $R(b, a)$ as a possible image

Independent Queries

$Q_1 = R(x, a), S(x, b)$ and $Q_2 = R(b, a), S(c, y)$ independent?

- both have $R(b, a)$ as a possible image

$Q_1 = R(x, a), S(x, b)$ and $Q_2 = R(x, b), S(x, d)$ independent?

Independent Queries

$Q_1 = R(x, a), S(x, b)$ and $Q_2 = R(b, a), S(c, y)$ independent?

- both have $R(b, a)$ as a possible image

$Q_1 = R(x, a), S(x, b)$ and $Q_2 = R(x, b), S(x, d)$ independent?

- no common image possible (why?)

Independent Queries

Proposition

If $Q_1, Q_2, \ldots, Q_k$ are syntactically independent queries, then $Q_1, Q_2, \ldots, Q_k$ are independent probabilistic events.

Independent Queries

Proposition

If $Q_1, Q_2, \ldots, Q_k$ *are syntactically independent queries, then* $Q_1, Q_2, \ldots, Q_k$ *are independent probabilistic events.*

- extensional query evaluation: apply simple probabilistic rules on independent parts of queries

Extensional Rules

Independent Join $\mathrm{P}(Q_1 \wedge Q_2) = \mathrm{P}(Q_1) \cdot \mathrm{P}(Q_2)$

Extensional Rules

Independent Join $\mathrm{P}(Q_1 \wedge Q_2) = \mathrm{P}(Q_1) \cdot \mathrm{P}(Q_2)$

Independent Union $\mathrm{P}(Q_1 \vee Q_2) = 1 - (1 - \mathrm{P}(Q_1))(1 - \mathrm{P}(Q_2))$

Extensional Rules

Independent Join $\mathrm{P}(Q_1 \wedge Q_2) = \mathrm{P}(Q_1) \cdot \mathrm{P}(Q_2)$

Independent Union $\mathrm{P}(Q_1 \vee Q_2) = 1 - (1 - \mathrm{P}(Q_1))(1 - \mathrm{P}(Q_2))$

Negation $\mathrm{P}(\neg Q) = 1 - \mathrm{P}(Q)$

Extensional Rules

For a query Q of the form $Q = \exists x.Q'$:

- x is a root variable if every atom $L \in Q'$ contains the variable x
- x is a separator variable if for any atoms L_1, L_2 that unify, x occurs in the same position

Extensional Rules

For a query Q of the form $Q = \exists x.Q'$:

- x is a root variable if every atom $L \in Q'$ contains the variable x
- x is a separator variable if for any atoms L_1, L_2 that unify, x occurs in the same position

For a query $Q = \exists x.Q'$ where x is a separator variable, then:

- for any two constants $a \neq b$, $Q'(a/x)$ and $Q'(b/x)$ are syntactically independent; and

Extensional Rules

For a query Q of the form $Q = \exists x.Q'$:

- x is a root variable if every atom $L \in Q'$ contains the variable x
- x is a separator variable if for any atoms L_1, L_2 that unify, x occurs in the same position

For a query $Q = \exists x.Q'$ where x is a separator variable, then:

- for any two constants $a \neq b$, $Q'(a/x)$ and $Q'(b/x)$ are syntactically independent; and
- **Independent Project** rule:

$$\mathrm{P} = 1 - \prod_{a \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(Q'\left(a/x\right)\right)\right)$$

Extensional Rules

Inclusion-Exclusion For a query $Q = Q_1 \wedge Q_2 \wedge \cdots \wedge Q_k$ ($Q_1, \ldots, Q_k$ not necessarily independent):

$$\mathrm{P}(Q) = -\sum_{s \subseteq [k], s \neq \emptyset} (-1)^{|s|} \mathrm{P}(\bigvee_{i \in [s]} Q_i)$$

Extensional Rules

Inclusion-Exclusion For a query $Q = Q_1 \wedge Q_2 \wedge \cdots \wedge Q_k$ ($Q_1, \ldots, Q_k$ not necessarily independent):

$$\mathrm{P}(Q) = -\sum_{s \subseteq [k], s \neq \emptyset} (-1)^{|s|} \mathrm{P}(\bigvee_{i \in [s]} Q_i)$$

$$\begin{aligned} \mathrm{P}(Q_1 \wedge Q_2) = {} & \mathrm{P}(Q_1) + \mathrm{P}(Q_2) \\ & - \mathrm{P}(Q_1 \vee Q_2) \end{aligned}$$

Extensional Rules

Inclusion-Exclusion For a query $Q = Q_1 \wedge Q_2 \wedge \cdots \wedge Q_k$ ($Q_1, \ldots, Q_k$ not necessarily independent):

$$\mathrm{P}(Q) = -\sum_{s \subseteq [k], s \neq \emptyset} (-1)^{|s|} \mathrm{P}(\bigvee_{i \in [s]} Q_i)$$

$$\begin{aligned} \mathrm{P}(Q_1 \wedge Q_2) = {} & \mathrm{P}(Q_1) + \mathrm{P}(Q_2) \\ & - \mathrm{P}(Q_1 \vee Q_2) \end{aligned}$$

$$\begin{aligned} \mathrm{P}(Q_1 \wedge Q_2 \wedge Q_3) = {} & \mathrm{P}(Q_1) + \mathrm{P}(Q_2) + \mathrm{P}(Q_3) \\ & - \mathrm{P}(Q_1 \vee Q_2) - \mathrm{P}(Q_1 \vee Q_3) - \mathrm{P}(Q_2 \vee Q_3) \\ & + \mathrm{P}(Q_1 \vee Q_2 \vee Q_3) \end{aligned}$$

Applying Extensional Rules: Algorithm

- start from Q and write $\mathrm{P}(Q)$ in terms of $\mathrm{P}(Q_1), \mathrm{P}(Q_2), \ldots$

Applying Extensional Rules: Algorithm

- start from Q and write $\mathrm{P}(Q)$ in terms of $\mathrm{P}(Q_1), \mathrm{P}(Q_2), \ldots$
- apply rules for each Q_i iteratively until we arrive at a ground tuple (for which we simply look up the probability)

Applying Extensional Rules: Algorithm

- start from Q and write $\mathrm{P}(Q)$ in terms of $\mathrm{P}(Q_1), \mathrm{P}(Q_2), \ldots$
- apply rules for each Q_i iteratively until we arrive at a ground tuple (for which we simply look up the probability)
- if the algorithm stops at ground tuples, then query is safe, otherwise unsafe

Applying Extensional Rules: Algorithm

- start from Q and write $\mathrm{P}(Q)$ in terms of $\mathrm{P}(Q_1), \mathrm{P}(Q_2), \ldots$
- apply rules for each Q_i iteratively until we arrive at a ground tuple (for which we simply look up the probability)
- if the algorithm stops at ground tuples, then query is safe, otherwise unsafe
- algorithm is non-deterministic, e.g., inclusion-exclusion

Applying Extensional Rules: Algorithm

- start from Q and write $\mathrm{P}(Q)$ in terms of $\mathrm{P}(Q_1), \mathrm{P}(Q_2), \ldots$
- apply rules for each Q_i iteratively until we arrive at a ground tuple (for which we simply look up the probability)
- if the algorithm stops at ground tuples, then query is safe, otherwise unsafe
- algorithm is non-deterministic, e.g., inclusion-exclusion
- if we can evaluate $\mathrm{P}(Q)$ using rules except inclusion-exclusion (exponential rewriting), then $\mathrm{P}(Q)$ is tractable

Simple Example

$$Q = R(x), S(x, y) = \exists x.(R(x) \wedge \exists y.S(x, y))$$

Simple Example

$$Q = R(x), S(x, y) = \exists x.(R(x) \wedge \exists y.S(x, y))$$

$$\mathrm{P}(Q) = 1 - \prod_{a \in \mathsf{ADom}} (1 - \mathrm{P}(R(a) \wedge \exists y.S(a, y)))$$

Simple Example

$$Q = R(x), S(x,y) = \exists x.(R(x) \wedge \exists y.S(x,y))$$

$$\begin{aligned}
\mathrm{P}(Q) &= 1 - \prod_{a \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(R(a) \wedge \exists y.S(a,y)\right)\right) \\
&= 1 - \prod_{a \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(R(a)\right) \cdot \mathrm{P}\left(\exists y.S(a,y)\right)\right)
\end{aligned}$$

Simple Example

$$Q = R(x), S(x,y) = \exists x.(R(x) \wedge \exists y.S(x,y))$$

$$\begin{aligned}
\mathrm{P}(Q) &= 1 - \prod_{a \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(R(a) \wedge \exists y.S(a,y)\right)\right) \\
&= 1 - \prod_{a \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(R(a)\right) \cdot \mathrm{P}\left(\exists y.S(a,y)\right)\right) \\
&= 1 - \prod_{a \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(R(a)\right) \cdot \prod_{b \in \mathsf{ADom}} \left(1 - \mathrm{P}\left(S(a,b)\right)\right)\right)
\end{aligned}$$

Exercise: Extensional Query

Booking

date	**teacher**	**room**
3011	C42	p_1
0712	C42	p_2
1412	C017	p_3
0401	C017	p_4

Room

room	**equipment**	
C42	projector	q_1
C42	none	$1 - q_1$
C017	projector	q_2
C017	none	$1 - q_2$

- calculus: $Q() : \; \exists d, r. B(d, r) \wedge R(r, \text{'none'})$

Table of contents

Extensional Operators

Extensional operator: standard relational algebra operator, extended to manipulate tuple probabilities

Extensional Operators

Extensional operator: standard relational algebra operator, extended to manipulate tuple probabilities

We want to extend relational database plans to probabilistic databases:

- any safe query has a safe plan (=plan which computes probabilities correctly)

Extensional Operators

Extensional operator: standard relational algebra operator, extended to manipulate tuple probabilities

We want to extend relational database plans to probabilistic databases:

- any safe query has a safe plan (=plan which computes probabilities correctly)
- can use it in "normal" DBMS to compute output probabilities (benefiting from optimization, parallelism, ...)

Extensional Operators

Extensional operator: standard relational algebra operator, extended to manipulate tuple probabilities

We want to extend relational database plans to probabilistic databases:

- any safe query has a safe plan (=plan which computes probabilities correctly)
- can use it in "normal" DBMS to compute output probabilities (benefiting from optimization, parallelism, ...)
- still can compute if queries are unsafe, but probabilities are incorrect (may be able to compute upper and lower bounds)

Extensional Operators

We work with tuple-independent databases:

Extensional Operators

We work with tuple-independent databases:

- each relation has schema of the form $R(A, p)$
- A regular attribute, p probability
- $\Pi_A(R)$ is the deterministic part
- we assume each tuple $a \in A$ is unique

Extensional Operators

We work with tuple-independent databases:

- each relation has schema of the form $R(A, p)$
- A regular attribute, p probability
- $\Pi_A(R)$ is the deterministic part
- we assume each tuple $a \in A$ is unique

Let us see how the operators of relational algebra $(\bowtie, \sigma, \pi, \cup)$ are implemented

Extensional Operators

Independent Join $\bowtie$

$$R \bowtie^{i}_{C} S = \{(a, b, p_R(a) \cdot p_S(b)) \mid a \in \Pi_A(R), b \in \Pi_B(S), (a, b) \in \Pi_A(R) \bowtie_C \Pi_B(S)\}$$

Extensional Operators

Independent Join $\bowtie$

$R \bowtie_C^i S = \{(a, b, p_R(a) \cdot p_S(b)) \mid a \in \Pi_A(R), b \in \Pi_B(S), (a, b) \in \Pi_A(R) \bowtie_C \Pi_B(S)\}$

Independent Project π – $u_1, \ldots, u_k$ are the attributes that have a common value a

$$\pi_a^i(R) = \left\{ \left(a, 1 - \prod_{u \in R : u.A = a} (1 - u.p) \right) \mid a \in \Pi_A(R) \right\}$$

Extensional Operators

Independent Union $\cup$ – for two relations $R(A_1, p)$, $S(A_2, p)$

$$R \cup^i_A S = \left\{ (a, 1 - (1 - p_R(a.A_1))\,(1 - p_S(a.A_2))) \mid a.A_1 \in \Pi_{A_1}(R) \vee a.A_2 \in \Pi_{A_2}(S) \right\}$$

Extensional Operators

Independent Union $\cup$ – for two relations $R(A_1, p)$, $S(A_2, p)$

$$R \cup^i_A S = \left\{(a, 1 - (1 - p_R(a.A_1))\,(1 - p_S(a.A_2))) \mid a.A_1 \in \Pi_{A_1}(R) \vee a.A_2 \in \Pi_{A_2}(S)\right\}$$

Selection σ

$$\sigma_C(R) = \{(a, p_R(a)) \mid C \models a\}$$

Extensional Operators

Independent Union $\cup$ – for two relations $R(A_1, p)$, $S(A_2, p)$

$$R \cup_A^i S = \left\{ (a, 1 - (1 - p_R(a.A_1))\,(1 - p_S(a.A_2))) \mid a.A_1 \in \Pi_{A_1}(R) \vee a.A_2 \in \Pi_{A_2}(S) \right\}$$

Selection σ

$$\sigma_C(R) = \{(a, p_R(a)) \mid C \models a\}$$

Complementation

$$C_A(R) = \left\{ (a, 1 - p_R(a)) \mid a \in \mathsf{ADom}(D)^k \right\}$$

Extensional Plan Example

Booking

teacher	room	
Antoine	C42	p_1
Antoine	C42	p_2
Silviu	C017	p_3
Silviu	C018	p_4

Room

room	equipment	
C42	projector	q_1
C017	projector	q_2
C018	projector	q_3

Who are the teachers teaching in rooms with projectors ?

Extensional Plan Example

Booking

teacher	room	
Antoine	C42	p_1
Antoine	C42	p_2
Silviu	C017	p_3
Silviu	C018	p_4

Room

room	equipment	
C42	projector	q_1
C017	projector	q_2
C018	projector	q_3

Who are the teachers teaching in rooms with projectors ?

- $\pi_{\text{teacher}}\left(B \bowtie \pi_{\text{room}}(\sigma_{\text{'projector'}}(R))\right)$

Extensional Plan Example

B

teacher	room	
Antoine	C42	p_1
Antoine	C42	p_2
Silviu	C017	p_3
Silviu	C018	p_4

$\pi_{\text{room}}(\sigma_{\text{'projector'}}(R))$

room	
C42	q_1
C017	q_2
C018	q_3

Who are the teachers teaching in rooms with projectors?

- $\pi_{\text{teacher}}\left(B \bowtie \pi_{\text{room}}(\sigma_{\text{'projector'}}(R))\right)$

Extensional Plan Example

$B \bowtie \pi_{\text{room}}(\sigma_{\text{'projector'}}(R))$

teacher	room	
Antoine	C42	$p_1 q_1$
Antoine	C42	$p_2 q_1$
Silviu	C017	$p_3 q_2$
Silviu	C018	$p_4 q_3$

Who are the teachers teaching in rooms with projectors?

- $\pi_{\text{teacher}}\left(B \bowtie \pi_{\text{room}}(\sigma_{\text{'projector'}}(R))\right)$

Extensional Plan Example

$\pi_{\text{teacher}}\left(B \bowtie \pi_{\text{room}}(\sigma_{\text{'projector'}}(R))\right)$

teacher	
Antoine	$1-(1-p_1q_1)(1-p_2q_1)$
Silviu	$1-(1-p_3q_2)(1-p_4q_3)$

Who are the teachers teaching in rooms with projectors?

- $\pi_{\text{teacher}}\left(B \bowtie \pi_{\text{room}}(\sigma_{\text{'projector'}}(R))\right)$

Plans For Unsafe Queries

If a query Q is unsafe, there does not exist a safe extensional plan.

Plans For Unsafe Queries

If a query Q is unsafe, there does not exist a safe extensional plan.

However, we can compute upper bounds in some cases:

Proposition

Let $Q = Q_1 \vee Q_2 \vee \cdots \vee Q_k$ and Q_i a conjunctive query without self-joins (no joins between the same relation names). Then any plan using independent join, indepedent projection, independent union and selection will compute an upper bound for the answer tuple probabilities.

Plans For Unsafe Queries

Query (unsafe) $Q : R(z, x), S(x, y), T(y)$

Plans For Unsafe Queries

Query (unsafe) $Q : R(z, x), S(x, y), T(y)$

$$
\begin{aligned}
P_1 =& \pi_z(\pi_{zx}(R(z, x) \bowtie S(x, y)) \bowtie T(y)) \\
P_2 =& \pi_z(R(z, x) \bowtie \pi_x(S(x, y) \bowtie T(y))) \\
P_3 =& \pi_z(R(z, x) \bowtie S(x, y) \bowtie T(y))
\end{aligned}
$$

Plans For Unsafe Queries

Query (unsafe) $Q : R(z,x), S(x,y), T(y)$

$$
\begin{aligned}
P_1 =& \pi_z(\pi_{zx}(R(z,x) \bowtie S(x,y)) \bowtie T(y)) \\
P_2 =& \pi_z(R(z,x) \bowtie \pi_x(S(x,y) \bowtie T(y))) \\
P_3 =& \pi_z(R(z,x) \bowtie S(x,y) \bowtie T(y))
\end{aligned}
$$

The above Proposition gives us a way to compute tighter upper bounds $\rightsquigarrow$ execute as many plans as possible an take the minimum as the estimated probability

Table of contents

Intensional Query Evaluation

Intensional Query Evaluation: compute probabilities of Q directly from the lineage formula Φ_Q

Intensional Query Evaluation

Intensional Query Evaluation: compute probabilities of Q directly from the lineage formula Φ_Q

- first, we compute the lineage of$Q$
- then, we compile lineage Φ_Q into a circuit allowing efficient evaluation

Intensional Query Evaluation

Intensional Query Evaluation: compute probabilities of Q directly from the lineage formula Φ_Q

- first, we compute the lineage of$Q$
- then, we compile lineage Φ_Q into a circuit allowing efficient evaluation

Ingredient – same concept of independence between two lineages:

- Q_1 and Q_2 are independent if they have *disjoint supports* $(\mathsf{Var}(Q_1) \cap \mathsf{Var}(Q_2) \neq \emptyset)$

Intensional Query Rules

Independent AND: $\mathrm{P}(\Phi_{Q_1 \wedge Q_2}) = \mathrm{P}(\Phi_{Q_1}) \cdot \mathrm{P}(\Phi_{Q_2})$

Intensional Query Rules

Independent AND: $\mathrm{P}(\Phi_{Q_1 \wedge Q_2}) = \mathrm{P}(\Phi_{Q_1}) \cdot \mathrm{P}(\Phi_{Q_2})$

Independent OR: $\mathrm{P}(\Phi_{Q_1 \vee Q_2}) = 1 - (1 - \mathrm{P}(\Phi_{Q_1}))(1 - \mathrm{P}(\Phi_{Q_2}))$

Intensional Query Rules

Independent AND: $\mathrm{P}(\Phi_{Q_1 \wedge Q_2}) = \mathrm{P}(\Phi_{Q_1}) \cdot \mathrm{P}(\Phi_{Q_2})$

Independent OR: $\mathrm{P}(\Phi_{Q_1 \vee Q_2}) = 1 - (1 - \mathrm{P}(\Phi_{Q_1}))(1 - \mathrm{P}(\Phi_{Q_2}))$

Negation: $\mathrm{P}(\neg\Phi_Q) = 1 - \mathrm{P}(\Phi_Q)$

Intensional Query Rules

Disjoint OR – two formulas Φ_1, Φ_2 are called disjoint if the formula $\Phi_1 \wedge \Phi_2$ is not satisfiable $\mathrm{P}(\Phi_1 \vee \Phi_2) = \mathrm{P}(\Phi_1) + \mathrm{P}(\Phi_2)$

Intensional Query Rules

Disjoint OR – two formulas Φ_1, Φ_2 are called disjoint if the formula $\Phi_1 \wedge \Phi_2$ is not satisfiable $\mathrm{P}(\Phi_1 \vee \Phi_2) = \mathrm{P}(\Phi_1) + \mathrm{P}(\Phi_2)$

Shannon expansion – a general rule; intuitively, choose a variable to instantiate and rewrite Φ

$$\mathrm{P}(\Phi) = \sum_{i=0,m} P(\Phi|_{x=a_i}) \cdot P(X = a_i)$$

Intensional Query Evaluation Using Rules

Algorithm:

- same as query evaluation in the extensional case: iteratively apply the rules until we arrive at ground tuples/variable;
- all rules require a form of independence, except Shannon expansion which can be applied anywhere

Intensional Query Evaluation Using Rules

Algorithm:

- same as query evaluation in the extensional case: iteratively apply the rules until we arrive at ground tuples/variable;
- all rules require a form of independence, except Shannon expansion which can be applied anywhere

Complexity:

- the algorithm is non-deterministic
- if only independent OR, AND, negation are applied, then the size of the probability formula is linear in Φ – size lower bound
- if only Shannon expansion can be used the formula is exponential in the size of Φ – size upper bound

Exercise: Lineage

$$Q() \iff x_1\neg y_1 \vee x_2\neg y_1 \vee x_3\neg y_2 \vee x_4\neg y_2$$

(assume $\mathrm{P}(x_i) = p_i$ and $\mathrm{P}(y_i) = q_i$)

Booking $\bowtie$ Room

date	**teacher**	**room**	**equipment**	
3011	Silviu	C42	projector	x_1y_1
3011	Silviu	C42	none	$x_1\neg y_1$
0712	Antoine	C42	projector	x_2y_1
0712	Antoine	C42	none	$x_2\neg y_1$
1412	Silviu	C017	projector	x_3y_2
1412	Silviu	C017	none	$x_3\neg y_2$
0401	Antoine	C017	projector	x_4y_2
0402	Antoine	C017	none	$x_4\neg y_2$

Read-Once Formulas

$$\Phi = x_1\neg y_1 \vee x_2\neg y_1 \vee x_3\neg y_2 \vee x_4\neg y_2$$
$$\Phi' = \neg y_1(x_1 \vee x_2) \vee \neg y_2(x_3 \vee x_4)$$

Read-Once Formulas

$$\Phi = x_1 \neg y_1 \vee x_2 \neg y_1 \vee x_3 \neg y_2 \vee x_4 \neg y_2$$
$$\Phi' = \neg y_1 (x_1 \vee x_2) \vee \neg y_2 (x_3 \vee x_4)$$

Definition

A formula ϕ is *read-once* iff there exists Φ' such that no variable is repeated more than once in it.

Read-Once Formulas

$$\Phi = x_1 \neg y_1 \vee x_2 \neg y_1 \vee x_3 \neg y_2 \vee x_4 \neg y_2$$
$$\Phi' = \neg y_1 (x_1 \vee x_2) \vee \neg y_2 (x_3 \vee x_4)$$

Definition

A formula ϕ is *read-once* iff there exists Φ' such that no variable is repeated more than once in it.

Efficient class of formulas:

- if read-once Φ' of Φ exists, it can be computed from Φ in polynomial times

Read-Once Formulas

$$\Phi = x_1 \neg y_1 \vee x_2 \neg y_1 \vee x_3 \neg y_2 \vee x_4 \neg y_2$$
$$\Phi' = \neg y_1 (x_1 \vee x_2) \vee \neg y_2 (x_3 \vee x_4)$$

Definition

A formula ϕ is *read-once* iff there exists Φ' such that no variable is repeated more than once in it.

Efficient class of formulas:

- if read-once Φ' of Φ exists, it can be computed from Φ in polynomial times
- $\mathrm{P}(\Phi')$, where Φ' read-once, can be computed in linear time by applying only independent AND, OR and negation

Table of contents

Circuits for Lineage Formulas

Compiling a formula Φ: converting into a Boolean circuit so that we can compute $\mathrm{P}(\Phi)$ efficiently.

Circuits for Lineage Formulas

Compiling a formula Φ: converting into a Boolean circuit so that we can compute $\mathrm{P}(\Phi)$ efficiently.

A circuit for Φ is a rooted, labeled DAG, containing a subset of the following types of gates:

1. *Independent AND*: labeled $\wedge$ having children variables or clauses,
2. *Independent/Disjoint OR* labeled $\vee$ having as children variables or clauses,
3. *NOT* labeled $\neg$, child the negated clause/variable
4. *Conditional gate* labeled with a variable X_i and having two edges corresponding to the clauses occurring when $X_i =$ true and $X_i =$ false (Shannon expansion),
5. *Leaf node*, either 1 or 0, or the variable X_i.

Circuits for Lineage Formulas

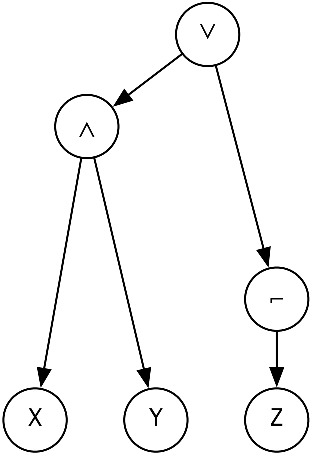
Compiling a formula Φ: converting into a Boolean circuit so that we can compute $\mathrm{P}(\Phi)$ efficiently.

A circuit for Φ is a rooted, labeled DAG, containing a subset of the following types of gates:

1. *Independent AND*: labeled $\wedge$ having children variables or clauses,
2. *Independent/Disjoint OR* labeled $\vee$ having as children variables or clauses,
3. *NOT* labeled $\neg$, child the negated clause/variable
4. *Conditional gate* labeled with a variable X_i and having two edges corresponding to the clauses occurring when $X_i = \text{true}$ and $X_i = \text{false}$ (Shannon expansion),
5. *Leaf node*, either 1 or 0, or the variable X_i.

Several possible compilation targets: read-once formulas, d-DNNF$^\neg$, OBDD, FBDD

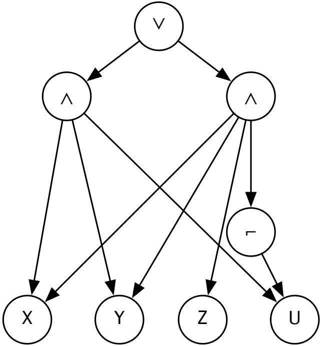
Read-Once Circuits



A read-once circuit contains only independent AND, independent OR, and NOT gates, and variables on leaf nodes.

Given a read-once circuit representing a formula Φ, one can compute $\mathrm{P}(\Phi)$in linear time. How?

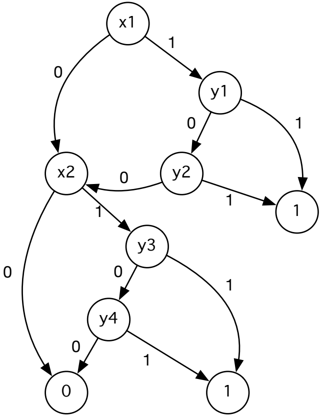
d-DNNF$^\neg$ (Deterministic Decomposable Negation Normal Form)



A d-DNNF$^\neg$ contains only independent AND, disjoint OR, and NOT gates, and variables on leaf nodes.

Given a d-DNNF$^\neg$ representing a formula Φ, one can compute $\mathrm{P}(\Phi)$in linear time. How?

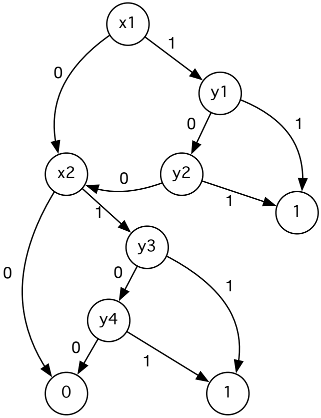
FBDD/OBDD (Free/Ordered Binary Decision Diagram)



A FBDD contains only conditional gates as nodes, and the leafs are either 1 or 0.

An OBDD is an FBDD with the property that all paths from the root to the leaves visit the nodes in the same order.

FBDD/OBDD (Free/Ordered Binary Decision Diagram)



A FBDD contains only conditional gates as nodes, and the leafs are either 1 or 0.
An OBDD is an FBDD with the property that all paths from the root to the leaves visit the nodes in the same order.

- a read-once expression Φ has an OBDD of linear size
- OBDDs can be build inductively on the structure of a formula Φ: $\Phi_1 \wedge \Phi_2$ or $\Phi_1 \vee \Phi_2$ have width[a] at most $w_1 w_2$

[a]width – the highest number of conditional gates

Exercise: Lineage Circuits

Compile $\Phi = x_1 y_1 \vee x_2 \neg y_1 \vee \neg x_2 \neg y_2 \vee \neg x_1 y_2$ into a d-DNNF$^\neg$.

Compile $\Phi = x_1 \neg y_1 \vee x_2 \neg y_1 \vee x_3 \neg y_2 \vee x_4 y_2$ into an OBDD and a read-once circuit.

Circuit Representations: Can We Always Use Them?

Computing a probability on a circuit is linear in its size – the size may still be exponential.

Circuit Representations: Can We Always Use Them?

Computing a probability on a circuit is linear in its size – the size may still be exponential.

Not all formulas can be represented efficiently with circuits. *What if we are happy with an estimated probability?*

Approximating Φ: Upper, Lower Bounds

We can apply the rules algorithm for upper and lower bounds, i.e., obtaining and interval $[L, U]$ such that $L \leqslant \mathrm{P}(\Phi) \leqslant U$

Approximating Φ: Upper, Lower Bounds

We can apply the rules algorithm for upper and lower bounds, i.e., obtaining and interval $[L, U]$ such that $L \leqslant \mathrm{P}(\Phi) \leqslant U$

Iteratively use formulas for $\wedge$ and $\vee$:

$$
\begin{aligned}
\max(\mathrm{P}(\Phi_1), \mathrm{P}(\Phi_2)) \leqslant \quad & \mathrm{P}(\Phi_1 \vee \Phi_2) \leqslant & \min(\mathrm{P}(\Phi_1) + \mathrm{P}(\Phi_2), 1) \\
\max(0, \mathrm{P}(\Phi_1) + \mathrm{P}(\Phi_2) - 1) \leqslant \quad & \mathrm{P}(\Phi_1 \wedge \Phi_2) \leqslant & \min(\mathrm{P}(\Phi_1), \mathrm{P}(\Phi_2)) \\
& \mathrm{P}(\neg\Phi) = & 1 - \mathrm{P}(\Phi)
\end{aligned}
$$

Approximating Φ: Monte-Carlo

We can use Monte-Carlo algorithms (sampling) for estimating any expression Φ , by repeating a sampling process N times:

1. choose a random valuation $\theta \in w(\Phi)$ proportionally to the probability $\mathrm{P}(\theta) \rightsquigarrow$ each variable X in Φ is set to 1 with probability $\mathrm{P}(X)$
2. if $\Phi(\theta)$ is true, then return $Z = 1$, otherwise $Z = 0$

Approximating Φ: Monte-Carlo

We can use Monte-Carlo algorithms (sampling) for estimating any expression Φ , by repeating a sampling process N times:

1. choose a random valuation $\theta \in w(\Phi)$ proportionally to the probability $\mathrm{P}(\theta) \rightsquigarrow$ each variable X in Φ is set to 1 with probability $\mathrm{P}(X)$
2. if $\Phi(\theta)$ is true, then return $Z = 1$, otherwise $Z = 0$

$\mathrm{P}(\Phi)$ is estimated as

$$\mathrm{P} = \frac{\sum_{k=1}^{N} Z_k}{N}$$

Approximating Φ: Monte-Carlo

We can use Monte-Carlo algorithms (sampling) for estimating any expression Φ , by repeating a sampling process N times:

1. choose a random valuation $\theta \in w(\Phi)$ proportionally to the probability $\mathrm{P}(\theta) \rightsquigarrow$ each variable X in Φ is set to 1 with probability $\mathrm{P}(X)$
2. if $\Phi(\theta)$ is true, then return $Z = 1$, otherwise $Z = 0$

$\mathrm{P}(\Phi)$ is estimated as

$$\mathrm{P} = \frac{\sum_{k=1}^{N} Z_k}{N}$$

We are looking for an (ϵ, δ)-approximation, *i.e.*:

$$\mathrm{P}(|\hat{p} - p| > \epsilon p) \leqslant \delta$$

Approximating Φ: Monte-Carlo

We can use Monte-Carlo algorithms (sampling) for estimating any expression Φ , by repeating a sampling process N times:

1. choose a random valuation $\theta \in w(\Phi)$ proportionally to the probability $\mathrm{P}(\theta) \rightsquigarrow$ each variable X in Φ is set to 1 with probability $\mathrm{P}(X)$
2. if $\Phi(\theta)$ is true, then return $Z = 1$, otherwise $Z = 0$

$\mathrm{P}(\Phi)$ is estimated as

$$\mathrm{P} = \frac{\sum_{k=1}^{N} Z_k}{N}$$

We are looking for an (ϵ, δ)-approximation, *i.e.*:

$$\mathrm{P}(|\hat{p} - p| > \epsilon p) \leqslant \delta$$

How many samples do we need? Use Chernoff bounds to get an estimation

$$N = \lceil \frac{4 \log \frac{2}{\delta}}{p\epsilon^2} \rceil$$

Approximating Φ

Monte-Carlo estimation may need an exponential number of samples to get a close estimation. Why?

Approximating Φ

Monte-Carlo estimation may need an exponential number of samples to get a close estimation. Why?

More advanced estimators can be used if we assume the lineage is represented in a certain way.

Example: if our formula is in **DNF** (Disjunctive Normal Form), then one can use the Karp-Luby estimator to efficiently compute probabilities.