The Inclusion-Exclusion Rule and its Application to the Junction Tree Algorithm

David Smith
Department of Computer Science
The University of Texas at Dallas
Richardson, TX, 75080, USA
dbs014200@utdallas.edu

Vibhav Gogate
Department of Computer Science
The University of Texas at Dallas
Richardson, TX, 75080, USA
vgogate@hlt.utdallas.edu

Abstract

In this paper, we consider the inclusion-exclusion rule – a known yet seldom used rule of probabilistic inference. Unlike the widely used sum rule which requires easy access to all joint probability values, the inclusion-exclusion rule requires easy access to several marginal probability values. We therefore develop a new representation of the joint distribution that is amenable to the inclusion-exclusion rule. We compare the relative strengths and weaknesses of the inclusion-exclusion rule with the sum rule and develop a hybrid rule called the inclusion-exclusion-sum (IES) rule, which combines their power. We apply the IES rule to junction trees, treating the latter as a target for knowledge compilation and show that in many cases it greatly reduces the time required to answer queries. Our experiments demonstrate the power of our approach. In particular, at query time, on several networks, our new scheme was an order of magnitude faster than the junction tree algorithm.

1 Introduction

Popular probabilistic inference algorithms such as bucket elimination [Dechter, 1999], recursive conditioning [Darwiche, 2001] and Belief propagation [Murphy *et al.*, 1999; Yedidia *et al.*, 2005] leverage only the sum and product rules of probability theory. In this paper, we consider the inclusion-exclusion rule, a known yet seldom used rule of probabilistic inference, and show that it presents several new opportunities for scaling up inference, specifically in the context of knowledge compilation [Darwiche and Marquis, 2002].

Knowledge compilation is a popular approach for tackling the intractability of probabilistic inference. The key idea is to compile the probabilistic graphical model (PGM) into a *tractable* structure such that several desired classes of queries can be answered in time that is polynomial (linear) in the size of the structure. The bulk of the computational overhead is thus pushed into the offline, compilation phase while queries can be answered quickly in an online manner. Tractable languages (e.g., junction or join trees [Lauritzen and Spiegelhalter, 1988; Dechter and Pearl, 1989], Arithmetic circuits [Darwiche, 2003], AND/OR multi-valued decision diagrams [Mateescu *et al.*, 2008], etc.) developed in the knowledge compilation community are also receiving increasing attention in the machine learning community. In particular, recently, there has been a push to learn tractable models directly from data instead of first learning a PGM and then compiling it into a tractable model (cf. [Bach and Jordan, 2001; Poon and Domingos, 2011]).

In this paper, we focus on *junction trees* (a tree of clusters of variables) – one of the oldest yet most widely used tractable subclass of PGMs – and seek to improve their query complexity. One of the main drawbacks of using junction trees as a target for knowledge compilation is that the time required to process each cluster is exponential in the number of variables in the cluster. Thus, when the cluster size is large (e.g., a few Gigabytes), even for easy queries, such as computing the marginal distribution over a subset of variables in a cluster, the method can be prohibitively expensive. We envision that such large junction trees will be stored on external storage devices such as hard disk drives or solid state drives [Kask *et al.*, 2010; Kyrola *et al.*, 2012]. Since disk I/O is much slower than RAM I/O, we want to minimize it. The method described in this paper will minimize disk I/O in many cases by allowing us to safely ignore large portions of each cluster, which in turn will reduce the query complexity.

We achieve this reduction in query complexity by considering alternate representations of potentials and developing efficient manipulation algorithms for these representations. Traditionally, each cluster potential in a junction tree is represented as a weighted truth table: each row in the table represents a truth assignment to all variables in the potential and is associated with a non-negative real number (weight). Although this representation enables efficient message passing algorithms, which require taking the product of potentials and summing out variables from potentials, it has a significant disadvantage. Inferring the probability of a conjunctive (probability of evidence) query defined over a subset of variables in a cluster is exponential in the number of variables not mentioned in the query and therefore queries that mention only a few variables are computationally expensive.

To remedy this problem, we propose an alternate representation. Given a truth assignment to all variables in a potential, considered as a set of literals (e.g., the assignment $A = 0$, $B = 1$ is the set $\{\neg a, b\}$), we represent the potential using the *power set of this set of literals*. Namely, we

represent the potential by attaching a weight to each subset of the set of literals. We develop an alternate inference rule, based the inclusion-exclusion principle, that uses this representation (we call it the inclusion-exclusion representation) to efficiently compute the probability of conjunctive queries. The main virtue of this new rule is that the probability of the conjunctive query can be computed in time that is exponential only in the number of literals in the query that are not present in the represented set of literals. Thus, when the number of query variables is small or when most literals in the query are present in the represented set of literals, the inclusion-exclusion rule is exponentially more efficient than the conventional approach.

However, a key drawback of the inclusion-exclusion representation is that computing products of potentials represented in the inclusion-exclusion format is very inefficient. As a result, we cannot use it for representing potentials in junction trees because the Belief propagation algorithm that is used to answer queries over the junction tree requires computing products of potentials. We therefore develop a hybrid representation (and a hybrid inference rule) which combines the power of the conventional truth-table representation with the inclusion-exclusion representation, and apply it to the junction tree algorithm. Our new algorithm takes variables that are exclusive to a cluster and instead of storing their assignments using a truth-table, uses the inclusion-exclusion approach to represent their most likely assignment. As a result, our most common queries have smaller complexity than our least common queries.

We evaluated the efficacy of our new approach, comparing it with the junction tree algorithm on several randomly generated and benchmark PGMs. Our results show that in many cases, at query time, our new algorithm was significantly faster than the junction tree algorithm, clearly demonstrating its power and promise.

2 The Sum Rule

Notation. Let $\mathcal{D}$ denote a discrete probability distribution defined over a set $\mathcal{X} = \{X_1, \ldots, X_n\}$ of binary variables that take values from the domain $\{\texttt{True}, \texttt{False}\}$. (We assume binary variables for convenience. Our method is general and can be easily extended to multi-valued variables.) A literal is a variable or its negation. We will denote literals of a variable by corresponding small letters. For example, a and $\neg a$ denote literals of A which are true iff A is assigned the value `True` and `False` respectively. A conjunctive feature is a conjunction of literals (e.g., a, $\neg a \wedge \neg b$, etc.). It is true when all of its literals are true and false otherwise. We will use the letters f, g, and q to denote the features; all other small letters denote literals. We will use Greek letters ϕ, ψ, etc., to denote functions and potentials.

To demonstrate our main idea, we will assume that the distribution $\mathcal{D}$ is specified using just one function or potential, denoted by ϕ. (Using just one (giant) potential to specify the joint distribution is clearly a bad idea. We will relax this requirement when we apply our approach to junction trees.) ϕ can be specified in a number of ways. The most widely used approach is to delineate all possible assignments to all variables in $\mathcal{X}$. Since each full assignment corresponds to a full conjunctive feature, we can also think of ϕ as a mapping between full conjunctive features and $\mathbb{R}^+$. For instance, a function over $\{A, B\}$ can be specified using $\phi : \{(\neg a \wedge \neg b), (\neg a \wedge b), (a \wedge \neg b), (a \wedge b)\} \rightarrow \mathbb{R}^+$. We will use the term *weight* to describe the real number associated with each conjunctive feature. For the rest of the paper, when ϕ is specified using full conjunctive features, we will say that ϕ is specified using the *sum-format*.

The probability distribution associated with ϕ is $P(f) = \phi(f)/Z$ where f is a full conjunctive feature and Z is the normalization constant, often called the partition function. Assuming that the partition function is known (computed offline and stored), we can infer the probability of any conjunctive query using the sum rule of probability theory:

$$\Pr(q) = \frac{1}{Z} \sum_{f \in \mathbb{F}(\mathcal{X} \setminus Vars(q))} \phi(q \wedge f)$$

where $Vars(q)$ denotes the set of variables mentioned in q and $\mathbb{F}(\mathcal{X} \setminus Vars(q))$ is the set of full conjunctive features of $\mathcal{X} \setminus Vars(q)$. We assume that q is non-trivial in that no two literals in q mention the same variable.

Example 1. Fig. 1(a) shows how the sum rule can be used to compute the probability of various queries.

Proposition 1. *Assuming that Z is known and the distribution is represented using full conjunctive features of $\mathcal{X}$, the time complexity of computing the probability of a conjunctive query containing k literals is $O(\exp(|\mathcal{X}| - k))$.*

Since the complexity of inference using the sum rule is exponential in $|\mathcal{X}| - k$, marginal queries over a small number of variables (i.e., k is small) are computationally expensive. In this paper, we argue that alternate descriptions of $\mathcal{D}$ can, in many cases, provide computationally cheaper alternatives.

3 The Inclusion-Exclusion (IE) Rule

Instead of describing ϕ in terms of weights attached to full conjunctive features over $\mathcal{X}$, we can describe it in terms of weights attached to all possible **positive conjunctive features** over $\mathcal{X}$ (we say that a conjunctive feature is positive if none of its literals are negated). For example, to describe a probability distribution over the set of random variables $\{A, B\}$, one would define the function $\delta : \{(a), (b), (a \wedge b), \bot\} \rightarrow \mathbb{R}^+$, where $\bot$ denotes the null feature and $\delta(\bot) = Z$. Such a description has the same size ($2^{|\mathcal{X}|}$) as the conventional description. Moreover, it completely specifies the distribution; the probability of any conjunctive query can still be retrieved, but now, instead of performing inference using the sum rule, it is more efficient to use the *inclusion-exclusion rule* [Knuth, 2005], which is based on the inclusion-exclusion principle. This is because many weights needed by the sum rule are not easily accessible in this alternative representation.

Formally, the inclusion-exclusion rule can be described as follows. Let δ be a mapping between positive conjunctive features over $\mathcal{X}$ and $\mathbb{R}^+$, let q be a conjunctive feature defined over some subset $\mathcal{E}$ of variables, let $\mathcal{E}_-$ be the set of variables associated with negative literals in q, let q_+ denote a conjunction of positive literals in q and let $\mathbb{P}(\mathcal{E}_-)$ be the set of all

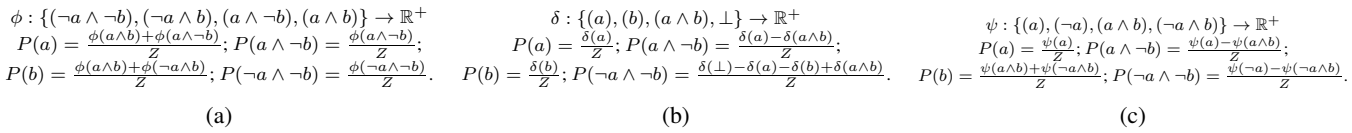


Figure 1: Demonstration of (a) the sum rule, (b) the inclusion-exclusion rule, (c) the hybrid inclusion-exclusion-sum rule. In the hybrid rule A is stored in sum-format and B is stored in IE-format.

possible positive conjunctive features over $\mathcal{E}_-$ (including the null feature). Then:

$$\Pr(q) = \frac{1}{Z} \sum_{f \in \mathbb{P}(\mathcal{E}_-)} (-1)^{|Vars(f)|} \delta(q_+ \wedge f)$$

For the rest of the paper, when δ is specified using positive conjunctive features, we will say that ϕ is specified using the *inclusion-exclusion-format* or *IE-format* in short.

Example 2. Fig. 1(b) demonstrates how to use the inclusion-exclusion rule to answer queries.

Proposition 2. *Assuming that Z is known and the distribution is represented using positive conjunctive features, the time complexity of computing the probability of a conjunctive query using the inclusion-exclusion rule is exponential in the number of negative literals in the query.*

Propositions 1 and 2 elucidate the trade-offs between the sum and inclusion-exclusion rules. For classes of queries involving a number of variables that is relatively small compared to the number of variables over which the distribution is defined, or for classes of queries that contain large number of positive literals, the inclusion-exclusion rule is much more efficient than the sum rule. Conversely, on classes of queries that contain large numbers of variables or negative literals, the sum rule is much more efficient.

There is no a priori reason, however, to describe the distribution using only positive conjunctive features and we assume this specification for convenience and simplicity. In principle, one can choose any full conjunctive feature and specify the distribution using the power set of the set of literals in the feature. For example, again suppose that we have a distribution over the random variables $\{A, B\}$, but now we have prior knowledge that most of the time our queries will ask about a and about $\neg b$. Then, we can specify the distribution using $\delta : \{(a), (\neg b), (a \wedge \neg b), \bot\} \to \mathbb{R}^+$. Now our most likely queries are more efficient than our less likely queries.

4 Hybrid Inclusion-Exclusion-Sum Rule

The discussion in the previous section lends itself readily to a hybrid approach that combines the power of inclusion-exclusion and sum rules. The key idea is to leverage prior knowledge about query types to achieve substantial reduction in complexity by switching between the two rules. For example, consider again our distribution over random variables $\{A, B\}$, but now suppose that we have prior knowledge that A will be commonly queried and both a and $\neg a$ are equally likely to appear in the query. Also, suppose that B will be rarely queried, but b is much more likely to appear in the query than $\neg b$. In this case, we can specify the distribution using $\psi : \{(a), (a \wedge b), (\neg a), (\neg a \wedge b)\} \to \mathbb{R}^+$, representing A using the sum-format and B using the IE-format. Again, our most common queries are now computationally easier than our less common queries. In this case however, we have to use the hybrid inclusion-exclusion-sum (IES) rule, which we describe formally next.

Let $\mathcal{V} \subseteq \mathcal{X}$ denote the set of variables stored in sum-format, let $\mathcal{E} = \mathcal{X} \setminus \mathcal{V}$ denote the set of variables stored in IE-format, let $\mathcal{E}_-$ be the subset of variables in $\mathcal{E}$ that appear negated in q, let q_{e+} be the conjunction of all positive literals in q projected on the set $\mathcal{E}$ and let q_v denote the conjunction of literals in q projected on the set $\mathcal{V}$. Then:

$$\Pr(q) = \frac{1}{Z} \sum_{f \in \mathbb{F}(\mathcal{V} \setminus Vars(q))} \sum_{g \in \mathbb{P}(\mathcal{E}_-)} (-1)^{|Vars(g)|} \delta(q_{e+} \wedge f \wedge g \wedge q_v)$$

Example 3. Fig. 1(c) demonstrates how to use the hybrid inclusion-exclusion-sum rule to answer queries.

Proposition 3. *Assuming that the distribution is represented using the IES format, the time complexity of computing the probability of a conjunctive query q using the IES rule is $O(\exp(|\mathcal{E}_-| + |\mathcal{V} \setminus Vars(q)|))$.*

The IES rule allows many more choices in how to represent a distribution. In particular, for a distribution with n random variables, we can represent k variables in the sum-format and then we have $2^{(n-k)}$ ways to choose a consistent assignment over the remaining variables (i.e., in the IE-format). So the total number of ways to represent the distribution is $\sum_{k=0}^{n} \binom{n}{k} 2^{n-k} = 3^n$, where the equality follows from the Binomial theorem. Each of these representations has its own computational advantages and disadvantages, and with some foreknowledge of the content queries, we can build distributions to leverage the advantages and avoid the disadvantages.

5 Hybrid IES Junction Trees

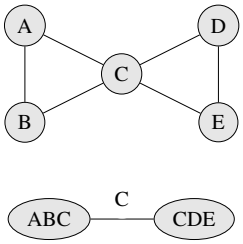
In the previous section, we showed that if we combine the inclusion-exclusion and sum rules, then there exist possibly several representations of a given probability distribution, each of which are equivalent in terms of size but differ in inference complexity at query time. However, distributions specified in such a fashion still require exponential space. Fortunately, in practice, one can leverage *conditional independence* to achieve compactness as well as tractability. For example, probabilistic graphical models exploit conditional independence to achieve compactness while junction trees use it to not only achieve compactness but also guarantee tractability of inference. In this section, we show how to apply the results from the previous section to junction trees, treating them as a target for knowledge compilation.

We begin with some required definitions.

A	B	C	$\phi_{A,B,C}$
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	1
1	0	1	2
1	1	0	3
1	1	1	4

C	D	E	$\phi_{C,D,E}$
0	0	0	1
0	0	1	2
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	0	1
1	1	1	2

(a)



(b)

	$\delta_{A,B,C}$
c	12
$\neg c$	8
$a \wedge c$	6
$a \wedge \neg c$	4
$b \wedge c$	8
$b \wedge \neg c$	6
$a \wedge b \wedge c$	4
$a \wedge b \wedge \neg c$	3

	$\delta_{C,D,E}$
c	6
$\neg c$	6
$c \wedge d$	3
$\neg c \wedge d$	3
$c \wedge e$	4
$\neg c \wedge e$	4
$c \wedge d \wedge e$	2
$\neg c \wedge d \wedge e$	2

(c)

Figure 2: (a) Two potentials defining a Markov network; (b) Primal graph and a possible junction tree for the Markov network given in (a); and (c) Potentials associated with the IES junction tree.

Definition 1. A **probabilistic graphical model** (PGM) or a **Markov network**, denoted by $\mathcal{M}$, is a pair $\langle \mathcal{X}, \Phi \rangle$ where $\mathcal{X}$ is a set of variables and Φ is a set of positive real-valued functions (potentials), each defined over a subset of variables. The probability distribution represented by $\mathcal{M}$ is given by $P_{\mathcal{M}}(f) = \prod_{\phi \in \Phi} \phi(f)/Z$ where f is a full conjunctive feature over $\mathcal{X}$ and Z is the partition function. The **primal graph** of $\mathcal{M}$ has variables of $\mathcal{M}$ as its vertices and an edge connects any two variables that appear in the same function.

Definition 2. Given a primal graph associated with a Markov network $\mathcal{M} = \langle \mathcal{X}, \Phi \rangle$, a **junction tree** is a tree $T(V, E)$ of labeled clusters that satisfies the following properties: (1) Each cluster $v \in V$ and each edge $e \in E$ is labeled with a subset of variables, denoted by $L(v)$ and $L(e)$ respectively; (2) Each variable is present in at least one cluster; (3) Both variables in each edge of the primal graph are present in at least one cluster; and (4) If a variable appears in two clusters, then it must be present in the labels of all edges on the unique path between the two clusters in T.

Example 4. Fig 2(a) shows two potentials that define a Markov network. Fig 2(b) shows the primal graph as well as a possible junction tree for the Markov network.

We can use junction trees for knowledge compilation in the following way (cf. [Dechter and Pearl, 1989; Shenoy and Shafer, 1990; Park and Darwiche, 2004]). In the offline compilation phase, we associate each cluster $v \in V$ in the junction tree with a function, denoted by ψ_v and initialize it to 1. Then, we multiply each function $\phi \in \Phi$ in the PGM with exactly one cluster function, say ψ_v, such that all variables in ϕ are present in v, storing the result in ψ_v. Finally, we *calibrate* the junction tree by selecting a cluster as the root and performing sum-product message passing in two passes: from the leaves to the root (collect pass) and then from the root to the leaves (distribute pass). Formally, the message sent from cluster u to cluster v, denoted by $m_{u \to v}$, is given by

$$m_{u \to v}(f) = \sum_{g \in \mathbb{F}(L(u) \setminus L(v))} \psi_u(f \wedge g) \prod_{w \in N(u); w \neq v} m_{w \to u}(f \wedge g) \tag{1}$$

where $N(u)$ is the set of neighbors of u in T.

The partition function of the Markov network can be computed at any cluster u using the following equation

$$Z = Z(u) = \sum_{f \in \mathbb{F}(L(u))} \psi_u(f) \prod_{v \in N(u)} m_{v \to u}(f) \tag{2}$$

Algorithm 1: Junction-Tree Query Answering

Input: A Calibrated Junction Tree $T(V, E)$ and a query q
Output: $\Pr(q)$
begin
  $T'(V', E') =$ Any rooted subtree of T containing all clusters in the set $\{v \mid L(v) \cap Vars(q) \neq \emptyset\}$
  Let $(e_1, \ldots, e_l)$ be the reverse depth-first ordering of edges from the leaves to the root of T'
  for *each* $v' \in V'$ **do**
    /* Set the weight of every feature in $\psi_{v'}$ that is inconsistent with q to zero */
    Instantiate-evidence($\psi_{v'}, q$)
  for $i = 1$ *to* l **do**
    Let $e_i = (u_i, v_i)$
    Update the message $m_{u_i \to v_i}$ using Eq. (1)
  return $Z(v_l)/Z$
  /* $Z(v_l)$ is computed using Eq. (2). */

Algorithm 1 outlines the procedure for answering queries in an online manner using a calibrated junction tree. The algorithm performs message passing only over the portion of the tree that is relevant to the query – the sub-tree T' that contains the variables $Vars(q)$ that constitute our query q. To perform message passing, the algorithm (a) instantiates evidence by projecting the query on each cluster function in T', (b) finds an appropriate (reverse depth-first) ordering of edges from the leaves to the root of T' and (c) updates the messages along the order. After message passing, the root node has enough information to calculate the probability of query, and the algorithm calculates and returns this probability value.

Next, we formally present the inclusion-exclusion-sum (IES) junction tree algorithm. The main idea in this algorithm is to represent each cluster in the calibrated junction tree using the hybrid IES representation and then at query time perform message passing by taking advantage of this representation using the hybrid IES rule. Before proceeding we introduce some additional notation: let $q \setminus P$ denote the feature obtained by deleting all literals of the variable P from q. For example, $(a \wedge b) \setminus A = b$ and $(\neg b \wedge c) \setminus B = c$.

Algorithm 2 shows how to convert a calibrated junction tree into a IES junction tree. Because computing a product of two functions requires mutually exclusive and exhaustive delineation of common variables, we propose to leave the separator variables in traditional sum-format, while convert-

Algorithm 2: Convert JT to IES-JT

Input: A Junction Tree $T(V, E)$
Output: A IES Junction Tree $T_S(V_S, E_S)$
begin
  $V_S = V$; $E_S = E$;
  for *each* $v \in V_S$ **do**
    $\delta_v = \psi_v$
    for *each variable* P *that is exclusive to* $L(v)$ **do**
      Replace every entry q in δ_v in which $\neg p$ appears by
      $\delta_v(q \setminus P) = \delta_v((q \setminus P) \wedge \neg p) + \delta_v((q \setminus P) \wedge p)$
  return $T_S(V_S, E_S)$

ing those variables that occur *exclusively in a cluster* into the subset format. The algorithm implements this proposal by replacing each entry q in the the original potential in which the exclusive variable, say P, appears negated by $q \setminus P$. $\delta(q \setminus P)$ is calculated by applying the sum rule, namely, $\delta_v(q \setminus P) = \delta_v((q \setminus P) \wedge \neg p) + \delta_v((q \setminus P) \wedge p)$. The operation requires $O(s \exp(n))$ time for each cluster, where n is the number of variables in the cluster and s is the number of variables that occur exclusively in that cluster, and hence need to be converted.

Example 5. Fig. 2(c) shows the result of applying Algorithm 2 to the potentials given in Fig. 2(a).

The algorithm for answering queries in IES junction trees is identical to Algorithm 1, with two differences (we exclude the pseudo-code for brevity). First, evidence is projected only on variables stored in sum-format. Namely, we first project the query on the variables stored in the sum-format. Let us denote this projected query on the cluster v by q_v. Then, we set to zero the weight of every feature in δ_v that is inconsistent with q_v. Second, in the message passing phase, we compute messages using the hybrid IES rule. Formally, the message $m_{u \to v}$ is computed using the following equations:

$$\chi_u(f \wedge q_+) = \sum_{g \in \mathbb{P}(\mathcal{E}_{u-})} (-1)^{|Vars(g)|} \delta_u(f \wedge q_+ \wedge g) \quad (3)$$

where $\mathcal{E}_u \subseteq L(u)$ is the set of variables that are exclusive to u, $\mathcal{E}_{u-}$ is the set of variables whose literals appear negated in q, and q_+ is the projection of the conjunction of positive literals in q on $\mathcal{E}_u$

$$m_{u \to v}(f) = \sum_{g \in \mathbb{F}(\mathcal{V}(u) \setminus L(v))} \chi_u(f \wedge g) \prod_{w \in N(u); w \neq v} m_{w \to u}(f \wedge g) \quad (4)$$

where $\mathcal{V}(u)$ is the set of variables stored in the sum-format in u and $\chi_u(f \wedge g)$ is given by Eq. (3).

Note that even though a junction tree and its equivalent IES junction tree have the same space complexity, the time complexity for the same probability of evidence query will, in general, be significantly different on each structure. For example, the full negative assignment to all variables in $\mathcal{X}$ has a complexity linear in the number of clusters on the junction tree (as does any other full assignment query), whereas it has complexity that is exponential in $max(|\mathcal{E}_v|)$ over each cluster v on the equivalent IES junction tree. Conversely, the probability of evidence query q, in which q is equal to the positive assignment to all separator variables in T, has complexity linear in the number of clusters on the IES junction tree, whereas the standard junction tree algorithm has complexity exponential in $max(|\mathcal{E}_v|)$ over all clusters $v \in V$.

6 Modifying the Junction Tree to take advantage of the IES rule

The IES-JT query answering algorithm performs well in those cases in which clusters in the junction tree contain large numbers of variables exclusive to a single cluster. Standard variable ordering heuristics (e.g., minfill) tend to produce junction trees with relatively few variables exclusive to any cluster. Therefore, it is often necessary to modify the junction trees (by merging clusters) output by the standard heuristics in order to take full advantage of the hybrid IES rule. Modifying the junction tree may increase its width. However, in some cases, the junction tree may yield substantial reductions in query complexity. For example,

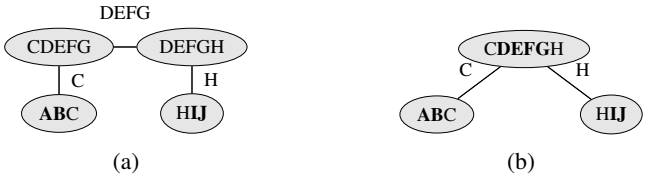


Figure 3: (a) A junction tree of width 5, and (b) a junction tree of width 6 obtained by merging clusters $CDEFG$ and $DEFGH$ of the junction tree in (a). Bold variables occur exclusively in a cluster.

Example 6. Consider the junction tree given in Figure 3a. The junction tree has a width of 5, but it has no variables exclusive to its maximum width clusters. Therefore, evidence queries that involve these maximum width clusters (e.g., $\Pr(a \wedge j)$) will still require performing a standard sum operation that is exponential in 5. Now consider the junction tree given in Figure 3b, which is obtained from the junction tree in Figure 3a by merging its two largest clusters. The new tree has a width of 6 and therefore its worst-case query complexity is exponential in 6. However, IES-JT query answering can now leverage the fact that variables D, E, F and G occur exclusively in a cluster. As a result, the query $\Pr(a \wedge j)$ will require time that is exponential in 2 rather than 6.

Algorithm 3 presents a greedy cluster merging procedure that takes as input a junction tree and modifies it so that is optimized for IES-JT query answering. The algorithm also takes as input a cluster bound C which limits the maximum cluster size. The cluster bound ensures that the worst-case query complexity as well as the space complexity of the junction tree is bounded by $\exp(max(C, width(T)))$. At each iteration, the algorithm greedily selects the variable X_i that yields the largest increase in the maximum number of exclusive variables in any cluster *without* (a) increasing the maximum number of non-exclusive variables in any cluster or (b) creating a cluster larger than the cluster bound C. This procedure is repeated until merging on every variable X_i either (a)

Algorithm 3: Merging Clusters

Input: A junction tree $T(V, E)$, an integer cluster bound C
Output: A modified Junction Tree
begin
  $Vars = \{L(v) | v \in V\}$; $maxscore = 1$;
  while $(Vars \neq \emptyset) \wedge (maxscore > 0)$ **do**
    $maxscore = 0$; $X_{max} = null$; $T_{max} = emptygraph$
    for *each* $X \in Vars$ **do**
      $V_T(X) = \{v \in V | X \in L(v)\}$
      /* Let $E(v)$ and $N(v)$ denote the set of variables in $L(v)$ that are exclusive and not exclusive to v respectively */
      $e = \max_{v \in V_T(X)} |E(v)|$
      $n = \max_{v \in V_T(X)} |N(v)|$
      $T' =$ Tree obtained from T by merging all clusters that mention X
      /* Let v' be the cluster in T' that contains X */
      if *(width*$(T') \leq C) \wedge$ *(*$|N(v')| \leq n$*)* **then**
        if $(|E(v')| - e) > maxscore$ **then**
          $maxscore = (|E(v')| - e)$
          $X_{max} = X$; $T_{max} = T'$
        else
          Remove X from $Vars$
    Remove X_{max} from $Vars$
    $T = T_{max}$
  return T

creates a tree whose merged cluster has a width larger than C, (b) increases the maximum number of non-exclusive variables in some cluster or (c) yields no increase in the number of variables exclusive to any cluster.

Note that Algorithm 3 is a naive, greedy approach to merging clusters. It will often yield junction trees that are far from optimal for IES-JT query answering. The development of an optimal merging algorithm is a priority of our future work.

7 Experiments

We conducted experiments on a number of randomly generated and benchmark PGMs, comparing the IES junction tree algorithm (IES-JT) with the conventional junction tree algorithm (JT). For lack of space, we only present a part of our results. We constructed the junction trees using the min-fill heuristic. For IES-JT, we then modified these junction trees using Algorithm 3.

7.1 Experiments on Chain Junction Trees

Our first set of results are on randomly generated chain junction trees with the cluster size k varying from 12 to 21. We randomly generated queries; each variable in the network participates in a query with probability p, where p ranges over $[0.1, 0.2, \ldots, 1.0]$. If a variable participates, it appears with a true assignment with probability 0.5. For each clique, we randomly generated potential weights; for each query probability we randomly generated 1000 queries. We measured the computation time of these queries using both the IES-JT and JT algorithms. The results are given in Figure 4.

p	k=12	k=15	k=18	k=21
0.1	**0.66** (20.74)	**1.91** (173.89)	**22.24** (1527.59)	**176.75** (11846.5)
0.2	**0.78** (14.07)	**1.78** (89.9)	**24.33** (726.67)	**203.82** (4449.33)
0.3	**0.85** (7.69)	**1.75** (40.08)	**20.03** (297.25)	**191.61** (1463.92)
0.4	**0.78** (4.41)	**1.78** (16.63)	**17.73** (92.53)	**150.61** (372.73)
0.5	**0.88** (2.31)	**1.78** (6.73)	**18.02** (33.65)	136.14 (**108.69**)
0.6	**0.89** (1.44)	**1.91** (2.99)	18.26 (**8.01**)	131.08 (**19.02**)
0.7	**0.95** (1.03)	2.14 (**1.48**)	18.94 (**2.69**)	132.47 (**4.93**)
0.8	1 (**0.8**)	2.49 (**0.93**)	19.59 (**1.41**)	138.22 (**1.61**)
0.9	1.05 (**0.73**)	2.73 (**0.78**)	21.82 (**1.04**)	145.36 (**1.1**)
1	1.09 (**0.65**)	3.18 (**0.76**)	25.61 (**0.94**)	154.69 (**0.88**)

Figure 4: Average query time (in ms) required by IES-JT and JT on chain junction trees having 10 clusters with cluster size $k \in \{12, 15, 18, 21\}$. In each column the time taken by JT is enclosed in parentheses while the one taken by IES-JT is not.

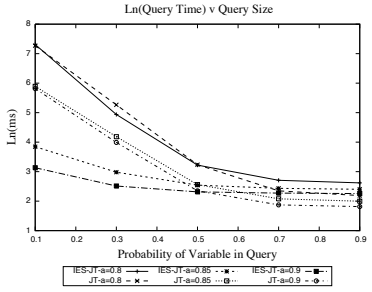


Figure 6: Log of average query time versus query size for random small world graphs.

The results show the strength of the IES-JT algorithm; as the clique size grows, JT suffers an exponential slowdown on queries over a small percentage of the variables, whereas IES-JT handles them extremely well. Conversely, JT outperforms IES-JT on queries involving large number of variables. However, the difference in performance remains relatively small, because on average only half of the literals in the query will be negated. Figure 5(a) visualizes the relationship for $k = 15$.

7.2 Experiments on small-world graphs

Our second method of problem generation is intended to model the structure of small-world networks. Such networks are characterized by the properties that (i) the average shortest path length is small compared to the size of the network, and yet (ii) the clustering coefficient (which measures degree to which nodes in a graph tend to cluster together) is large [Watts and Strogatz, 1998]. They are known to arise naturally from a wide variety of phenomena including social networks, Internet connectivity, road maps and network of neurons.

We generated small-world, power-law graphs using the following procedure. The procedure takes as input three integers p, r_1, and r_2, and a real number $a \in (0, 1]$ where p is the number of potentials, r_1 is a soft-bound on the potential size, and r_2 is a soft-bound on connectivity between potentials. We begin with a single potential with one variable. We then iteratively add p potentials to the graph. At each step, we add n new variables to the new potential as determined by the power law $n = \lceil r_1 \times x^{-log_a(r_1)} \rceil$, in which x is generated uniformly at random from the real-interval (0,1]. Each new variable is added to $k = \lceil r_2 \times x^{-log_a(r_2)} \rceil$ existing potentials. We evalu-

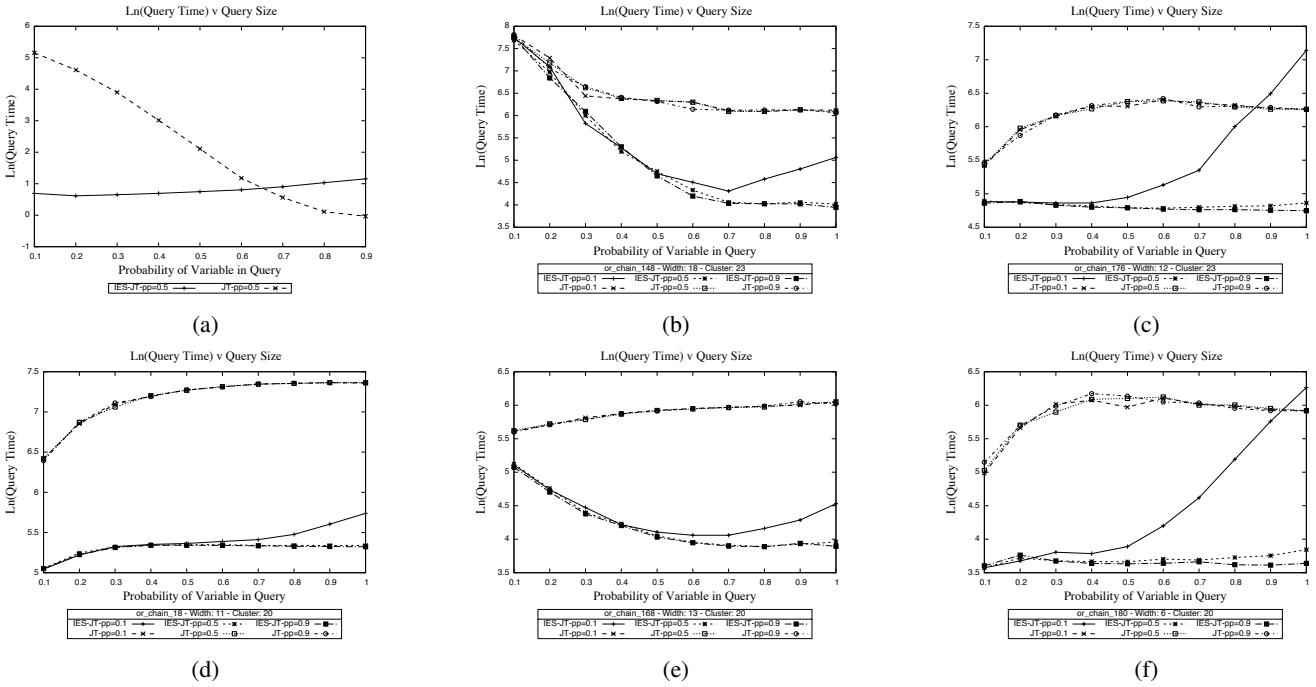


Figure 5: (a) Log of average query time versus query size on a chain junction tree having 10 clusters with the cluster size k bounded by 15; (b–f) Log of average query time versus query size for various Promedas networks. For the Promedas networks, each plot caption shows the width of the junction tree used as well as the "cluster bound" used in Algorithm 3 to construct the IES-JT.

ated our algorithms for large values for a, because these lead to larger clustering coefficients. We generated queries using the same procedure outlined in the previous subsection.

Figure 6 shows the average query times as a function of the query size for randomly generated power-law graphs with parameters $p = 50$, $r_1 = 20$, $r_2 = 3$, and $a \in \{0.8, 0.85, 0.90\}$. For each value of a, we generated 100 random networks and 1000 queries. IES-JT significantly outperforms JT when the query size is small, and JT outperforms IES-JT, but only marginally, when the query size is large.

7.3 Experiments on medical diagnosis networks

We also compared the JT and IES-JT algorithms on the Promedas medical diagnosis networks [Wemmenhove *et al.*, 2008] (freely available from the UAI 2008 evaluation repository). For lack of space, we only report on results for five benchmark networks `or_chain_148.fg.uai`, `or_chain_176.fg.uai`, `or_chain_18.fg.uai`, `or_chain_168.fg.uai`, and `or_chain_180.fg.uai`. For each network, we randomly generated queries using the following approach: each variable in the network participates in the query with probability pm, where $pm \in \{0.1, 0.2, \ldots, 1.0\}$, and is assigned the value true with probability pp, where $pp \in \{0.1, 0.2, \ldots, 1.0\}$.

Figures 5b through 5f show the results for various values of pp and pm. We observe that IES-JT is superior to JT when either pm is small or pp is large. As expected, JT tends to outperform IES-JT in those cases where pm is large and pp is small. IES-JT performs extremely well in those cases when the clustering algorithm successfully merges the largest clusters in the associated join-tree, but it is worth noting that even in cases where it does not (e.g., Figure 5b) IES-JT can still offer performance gains because queries on its modified junction tree require fewer total products.

8 Summary

In this paper, we showed how the inclusion-exclusion rule and its combination with the widely used sum rule can be used to reduce the query complexity in junction trees, when the latter is used as a target for knowledge compilation. The key idea in our approach is to leverage prior knowledge about query types to yield a tractable model that admits fast inference for the most common queries at the expense of admitting slightly slower inference for the least common queries. Our experimental analysis was preliminary but clearly demonstrated the power and promise of our approach.

Directions for future research include: developing efficient algorithms for merging clusters (cf. [Kask *et al.*, 2005]) trading time with space; applying the hybrid IES rule to other tractable models such as arithmetic circuits [Darwiche, 2003]; learning IES junction trees directly from data; using external memory to store junction trees; using the IES rule in generalized Belief propagation; etc.

Acknowledgements

This research was partly funded by the ARO MURI grant W911NF-08-1-0242. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO or the U.S. Government.

References

[Bach and Jordan, 2001] F. R. Bach and M. I. Jordan. Thin Junction Trees. In *Advances in Neural Information Processing Systems*, pages 569–576. MIT Press, 2001.

[Darwiche and Marquis, 2002] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[Darwiche, 2001] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126:5–41, February 2001.

[Darwiche, 2003] A. Darwiche. A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM*, 50:280–305, 2003.

[Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[Kask *et al.*, 2005] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1):165–193, 2005.

[Kask *et al.*, 2010] K. Kask, R. Dechter, and A. Gelfand. BEEM : Bucket elimination with external memory. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 268–276, Corvallis, Oregon, 2010. AUAI Press.

[Knuth, 2005] K. H. Knuth. Lattice duality: The origin of probability and entropy. *Neurocomputing*, 67:245–274, 2005.

[Kyrola *et al.*, 2012] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, Hollywood, October 2012.

[Lauritzen and Spiegelhalter, 1988] S. L. Lauritzen and D. J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[Mateescu *et al.*, 2008] R. Mateescu, R. Dechter, and R. Marinescu. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. *Journal of Artificial Intelligence Research*, 33:465–519, 2008.

[Murphy *et al.*, 1999] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[Park and Darwiche, 2004] J. D. Park and A. Darwiche. A differential semantics for jointree algorithms. *Artificial Intelligence*, 156(2):197 – 216, 2004.

[Poon and Domingos, 2011] H. Poon and P. Domingos. Sum-Product Networks: A New Deep Architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 337–346. AUAI Press, 2011.

[Shenoy and Shafer, 1990] P. Shenoy and G. Shafer. Valuation-based systems for discrete optimization. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 334–343, Corvallis, Oregon, 1990. AUAI Press.

[Watts and Strogatz, 1998] D. Watts and S. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998.

[Wemmenhove *et al.*, 2008] B. Wemmenhove, J. Mooij, W. Wiegerinck, M. Leisink, and H. J. Kappen. Inference in the promedas medical expert system. In *Proceedings of the 11th Conference on Artificial Intelligence in Medicine (AIME 2007)*. Springer, July 2008.

[Yedidia *et al.*, 2005] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized Belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.