Efficient Reasoning

Russell Greiner
and
Christian Darken
and
N. Iwan Santoso

Many tasks require "reasoning" — *i.e.*, deriving conclusions from a corpus of explicitly stored information — to solve their range of problems. An ideal reasoning system would produce all-and-only the *correct* answers to every possible query, produce answers that are as *specific* as possible, be *expressive* enough to permit any possible fact to be stored and any possible query to be asked, and be (time) *efficient*. Unfortunately, this is provably impossible: as correct and precise systems become more expressive, they can become increasingly inefficient, or even undecidable. This survey first formalizes these hardness results, in the context of both logic- and probability-based reasoning, then overviews the techniques now used to address, or at least side-step, this dilemma.

Categories and Subject Descriptors: I.2.3 [**Computing Methodologies**]: Artificial Intelligence—*Deduction and Theorem Proving — Answer/reason extraction, Inference Engines, Probabilistic Reasoning*; I.2.4 [**Computing Methodologies**]: Artificial Intelligence— *Knowledge Representation Formalisms and Methods — Bayesian Belief Nets, Rule-based Systems*

General Terms: Performance, Algorithms

Additional Key Words and Phrases: Efficiency Tradeoffs, Soundness/Completeness/Expressibility

---

1. INTRODUCTION

Many information systems use a corpus of explicitly stored information (a.k.a. a "knowledge base", KB) to solve their range of problems. For example, medical diagnostic systems use general facts about diseases, as well as the specific details of a particular patient, to determine which diseases the patient might have, and which treatment is appropriate. Similarly, configuration and synthesis systems use

---

Name: R Greiner
Address: Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada
Name: C Darken and N I Santoso
Address: Adaptive Information and Signal Processing, Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540, USA

their stored descriptions of various components, along with the specifications for a proposed device (VLSI chip, software program, factory, or whatever), to design a device that satisfies those requirements. Scheduling and planning systems likewise synthesize schedules sufficient to achieve some specified objective.

In each case, the underlying system must **reason** — that is, derive conclusions (*e.g.*, diagnoses, designs, schedules) that are sanctioned by its knowledge. Typically, an expert first, at "compile time", provides a corpus of general background facts — about medicine, or types of components, etc. At "run time", a user then specifies the details of a specific situation (*e.g.*, the symptoms of a specific patient, or the specifications of a particular desired artifact), and then poses some specific questions (Which disease? What should be connected to what? ...); the reasoning system then produces appropriate answers, based on its current KB which includes both the specifics of this problem and the general background knowledge. As there is a potentially infinite set of possible situations, these conclusions are typically not explicitly stored in the KB, but instead are computed as needed. This computation is called "reasoning" (aka "derivation", "deduction", "inference").

In general, we will identify a reasoner with its symbolic knowledge base KB; the user can pose queries χ to that reasoner and receive answers — *e.g.*, that χ is true or not. Section 2 motivates the use of such symbolic knowledge-based reasoners, and presents broad categories of such systems: logic-based (typically using Horn clauses) and probabilitic (using Bayesian belief nets). It also argues that we should evaluate a reasoner based on its facility in answering queries, using as quality measures: correctness, precision, expressiveness and efficiency.

We clearly prefer a reasoner that always returns all-and-only the correct and precise answers, immediately, to arbitrary queries. Unfortunately, we will see that this is not always possible (Section 2.4). Many implemented reasoning systems, therefore, sacrifice something — correctness, precision or expressiveness — to gain efficiency. The remainder of this paper presents various approaches: Section 3 (resp., Section 4, Section 5) overviews ways of improving worst-case efficiency by reducing *expressiveness* (resp., by allowing *imprecise* answers, by allowing occasional *incorrect* responses). Section 6 considers ways of producing (expressive, precise and correct) systems whose *"average-case"* efficiency is as high as possible. It also discusses ways to produce a system with high average *performance*, where the "performance" measure is a combination of these various criteria. Appendix A provides additional relevant details about belief nets.

2. SYMBOLIC REASONERS

2.1 Why Symbolic Reasoners?

In general, the user will interact with a knowledge-based *symbolic reasoner* via two subroutines: `Tell(`KB`, `χ`)` which tells the reasoner to extend its knowledge base KB to include the new information χ; and `Ask(`KB`, `χ`)` which asks the reasoner whether χ is true — here the reasoner's answer will often convey other information (such as a binding, or a probability value) to the user [Levesque 1984].[1]

[1]To be completely general, we may also have to include routines that *retract* some assertions, or in general revise our beliefs [Alchourrón et al. 1985]; we will not consider this issue here.

The underlying knowledge base is "symbolic", in that each of its individual components, in isolation, has "semantic content" — *e.g.*, the KB may contain statements about the world, perhaps in propositional logic, predicate calculus, or some probabilistic structure (details below). (This is in contrast to, for example, knowledge encoded as a numeric function, perhaps embodied in a neural net.[2])

There are many reasons why such a symbolic encoding is useful:

Explanation: Many systems have to interact with people. It is often crucial that these systems be able to justify why they produced their answers (in addition, of course, to supplying the correct answer). Many symbolic reasoners can convey their decisions, and justifications, to a person using the terms they used for their computations; as these terms have semantics, they are typically meaningful to that user.

Construction (as well as debugging and maintaining): As the individual components of the system (*e.g.*, rules, random variables, conditional probabilities, ...) are meaningful to people in isolation, it is relatively easy for an expert to encode the relevant features. This semantics also helps an expert to "debug", or update, a problematic knowledge base — here again a domain expert can examine a single specific component, in isolation, to determine whether it is appropriate.

Handling partial information — `or`, `not`, `exists`, as well as distributions: Many formalisms, including both ones that admit explicit reasoning and others, are capable of dealing with complete knowledge, corresponding to conjuctions and universally quantified statements — *e.g.*, "`gender = female and disease = meningitis`"; or "`everyone with meningitis is jaundiced`". In many situations, however, we may only have *partial* knowledge: "`disease is either meningitis or hepatitis`"; or "`some people with meningitis are not jaundiced`". Most logics, including any containing propositional logic, can readily express this information, using disjunction and negation ("`or`"s and "`not`"s). Yet more expressive systems, such as predicate calculus, can also deal with existentials. We may also want to explicitly state how confident we are of some claim, perhaps using probabilities.

2.2 Broad Categories of Reasoning Systems

There are many standard formalisms for encoding semantic information, each with its associated type of reasoning. This report will consider the following two major categories.[3]

1. (Sound and Monotonic) Logical Reasoning: This formalism assumes we have precise discrete descriptions of objects, and that we expect to obtain precise categorical answers to the questions posed. In particular, these systems can provide

[2] In some neural net systems, like KBANN [Towell and Shavlik 1993], the *nodes* do have semantic content, in that they refer to some event in the real world. The link-weights, however, are not semantic — their values are set only to provide accurate predictions. In general, there is no way to determine whether the weight on the $A \mapsto B$ link should be 0.7 vs 0.8 vs 0.0001, except in reference to the other weights, and with respect to some set of specific queries.

[3] As many readers may not be familiar with the second category. this report will provide more details here; see also Appendix A.

assurances that its answers will be "correct" — that is, if you believe the input facts, then you have to believe the conclusions.

In more detail: This category includes propositional logic and first order logic (predicate calculus), as well as higher order logics [Genesereth and Nilsson 1987; Enderton 1972; Chang and Lee 1973]. As an example, a logical system may include facts like "`Any woman of child-bearing age, with a distended abdomen, is pregnant.`". If we then assert that a particular patient is `female`, is of the correct `age`, and has a `distended abdomen`, the reasoner can then conclude that she is `pregnant`.

This is because the known facts

$$\mathcal{F} \quad = \quad \left\{ \begin{array}{l} \forall x \ \texttt{Woman}(x) \ \wedge \ \texttt{CBA}(x) \ \wedge \ \texttt{Dis\_Abd}(x) \implies \texttt{Pregnant}(x) \\ \texttt{Woman(Wilma)} \\ \texttt{CBA(Wilma)} \\ \texttt{Dis\_Abd(Wilma)} \end{array} \right\}$$

logically entails `Pregnant(Wilma)`, written $\mathcal{F} \models$ `Pregnant(Wilma)`. A reasoning process $\vdash_\alpha$ is "correct" (here, aka "sound") if, for any sets of propositions Φ and Σ,

$$\Phi \vdash_\alpha \Sigma \qquad \Rightarrow \qquad \Phi \models \Sigma$$

that is, $\vdash_\alpha$ only allows a reasoner to conclude "true" facts.

Typical logic-based systems use a collection of "inference rules" (possibly augmented with rewrite rules) to infer new statements from an existing KB. If each of these rules is sound (read "truth preserving"), then the resulting extended KB' will be as correct as the initial KB. In 1965, Robinson [Robinson 1965] proved that one can word any logical (first order) inference in terms of resolution. The ongoing challenge has been to find this proof as efficiently as possible — see Section 2.3.[4]

There are also many systems that use other sound deductive techniques, such as natural deduction (THINKER [Pelletier 1986], MIZAR [Rudnicki 1992] and ONTIC [McAllester 1989]) and/or equational logic (OTTER [McCune and Wos 1997]). Note that most of these systems also use resolution, in addition to their more specialized inference rules. Also, many of these systems are better viewed as "proof-checkers" rather than "theorem provers", as their main function is to verify that a proposed proof is legitimate. Hence, they hope to gain efficiency by sharing the burden with a (hopefully insightful) person. For brevity, this report will not further discuss such approaches.

Non-monotonic Logical Reasoning: Standard logical inference is *monotonic*, in that new information will never cause the reasoner to retract any conclusion. For example, after deriving that "`Wilman is pregnant`" from our general medical information together with information specific to `Wilman`, finding new information will not change this conclusion. This is not always appropriate, as subsequent information can compromise prior conclusions. (Imagine, for example, finding that `Wilma` had a hysterectomy.)

[4]See [Genesereth and Nilsson 1987] for a general discussion of these ideas — inference rules, soundness, resolution.

J	B	H	$P(J, B, H)$
0	0	0	0.03395
0	0	1	0.0095
0	1	0	0.0003
0	1	1	0.1805
1	0	0	0.01455
1	0	1	0.038
1	1	0	0.00045
1	1	1	0.722

Table 1. Joint Probability Distribution

Note that this does not mean the earlier conclusion was inappropriate; given that earlier store of knowledge, it was probably the correct interpretation. It is useful, however, to allow the reasoner to change its mind, given new data.

This intuitive concept of "nonmonotonicity" is well-motivated, and seems essential to common-sense reasoning. Where there is now a vast literature on this topic, including work on default reasoning [Reiter 1987], circumscription [McCarthy 1980], autoepistemic logics [Marek and Truszczyński 1989], as well as many variants (see also [Bobrow 1980] and [Ginsberg 1987]), it has proven very difficult to provide an effective implementation.[5] The major problem is understanding how to *represent* and *use* defeasible statments. That is, while we know how to deal with statement of the form "All birds fly", it is not so clear how to deal with the claim that "A bird, by default, flies": here, we *do* want to conclude initially that the bird `Tweety` will fly, but then reverse this conclusion later, on finding that `Tweety` is a penguin, or has a broken wing, or ...

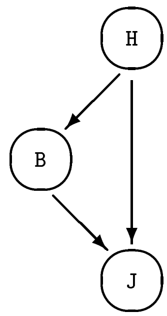
Fortunately, there are alternative approaches, which have precise definitions as well as implementations; see the decision-theoretic system mentioned below. This survey will therefore not further discuss "non-monotonic reasoning" formalisms.

2. Probabilistic Reasoning: Many forms of information are inherently probabilistic — eg, given certain symptoms, we may be 80% confident the patient has hepatitis, or given some evidence, we may be 10% sure a specific stock will go up in price.

One possible downside of dealing with probabilities is the amount of information that has to be encoded: in general one may have to express the entire joint distribution, which is exponential in the number of features; see Table 1. For many years, this observation motivated researchers to seek ways to avoid dealing with probabilities.

In many situations, however, there can be more laconic ways to express such information, by "factoring" the joint distribution. This has led to "Belief Nets" (aka "causal nets", "probability networks", "Bayesian nets"), which over the last decade have become the representation of choice for dealing with uncertainty [Pearl 1988; Shafer and Pearl 1990; Charniak 1991].

[5] The most visible implementation is "Theorist" [Poole et al. 1987], which handles a (useful and intuitive) subset of default reasoning.



$P($ H=1 $)$	$P($ H=0 $)$
0.05	0.95

H	$P($ B=1 | H $)$	$P($ B=0 | H $)$
1	0.95	0.05
0	0.03	0.97

H	B	$P($ J=1 | H, B $)$	$P($ J=0 | H, B $)$
1	1	0.8	0.2
1	0	0.8	0.2
0	1	0.3	0.7
0	0	0.3	0.7

Fig. 1. Simple (but not minimal) Belief Network

To make this concrete, consider the claims that Hepatitis "causes" Jaundice and also "causes" a Bloodtest to be positive, in that the chance of these symptoms will increase if the patient has hepatitis. We can represent this information using the full joint over these three binary variables (see Table 1 for realistic, if fabricated, numbers), then use this information to compute, for example, $P(\, h \,|\, \neg b\,)$ — the posterior probability that a patient has hepatitis, given that he has a negative blood test.[6] The associated computation,

$$P(\, h \,|\, \neg b\,) \quad = \quad \frac{P(\, h, \neg b\,)}{P(\, \neg b\,)} \quad = \quad \frac{\left(\sum_{X \not\in \{H,B\}} \sum_{x \in X} P(\, h, \neg b,\ X = x\,)\right)}{\left(\sum_{X \not\in \{B\}} \sum_{x \in X} P(\, \neg b,\ X = x\,)\right)}$$

involves the standard steps of *marginalization* (the summations shown above) to deal with unspecified values of various symptoms, and *conditionalization* (the division) to compute the conditional probability; see [Feller 1966]. In general, we will ask for the distribution of the "query variable" (here H) give the evidence specified by the "evidence variables" (here $B = \neg b$).

A Belief Network would represent this as a graphical structure, whose nodes represent probabilistic variables (such as "Hepatitis"), and whose directed links, roughly speaking, represent "causal dependencies", with the understanding that there should be a directed path from A to B, possibly a direct connection, if knowing the value of A can help specify the value for B. In particular, each node B has an associated "Conditional Probability Table" (aka "CPTable") that specifies the conditional distribution for B, given every assignment to B's parents; see Figure 1.

(For general binary-valued variables, the CPtable for a node X with k parents $\{\mathtt{Y}_i\}_{i=1}^k$ will include 2^k rows — one for each of the 2^k possible assignments to $\vec{Y} = \langle\, \mathtt{Y}_1, \ldots, \mathtt{Y}_k\,\rangle$ — and 2 columns, one for each possible value for X. Here, the $\langle\, i, j\,\rangle$ entry in this table will specify the conditional probability $P(\, X = i \,|\, \vec{\mathtt{Y}} = \vec{\mathtt{j}}\,)$ where $\vec{\mathtt{j}}$ represents the j^{th} assignment to $\vec{Y}$ (here $\vec{\mathtt{j}} \in \{\langle\, 0, \ldots, 0, 0\,\rangle,\ \langle\, 0, \ldots, 0, 1\,\rangle,\ \cdots,\ \langle\, 1, \ldots, 1\,\rangle\}$). Note the final column of the table is superfluous as each row must add up to 1. Of

[6] Note we are identifying a node with the associated variable. We will also use lower-case letters for values of the (upper case) variables; hence $H = h$ means the variable H has the value h. We will sometimes abbreviate $P(\, H = h\,)$ as $P(\, h\,)$. Finally, "$\neg h$" corresponds to $h = 0$, and "h" to $h = 1$.

course, these ideas generalize to general ℓ-ary variables. There is a similar approach when dealing with *continuous* variables [Pearl 1988].)

Notice, however, there is some "redundancy" in the network shown in Figure 1: given that the patient α has hepatitis, the probability that α has jaundice does *not* depend on whether α had a positive blood test — *i.e.*, $P(\texttt{J=1} \mid \texttt{H=1},\ \underline{\texttt{B=1}}) = 0.8 = P(\texttt{J=1} \mid \texttt{H=1},\ \underline{\texttt{B=0}})$ and $P(\texttt{J=1} \mid \texttt{H=0},\ \underline{\texttt{B=1}}) = 0.3 = P(\texttt{J=1} \mid \texttt{H=0},\ \underline{\texttt{B=0}})$. This means that jaundice is independent of blood test, *given hepatitis*.

This reduction — called "factoring" — allows us to use a simpler network, shown in Figure 2.[7] These factored representations include only the "relevant" connections: only include $A \mapsto B$ if A can directly influence B. This means the resulting network typically requires the user to specify fewer links, and fewer parameters (CPtable entries), and means the inference process (discussed in Appendix A.2) can generally be performed more efficiently. While the saving here is relatively small (2 links rather than 3, and a total of 5 parameters, rather than 7), the savings can be very significant for larger networks. As a real-world example, the complete joint distribution for the `Alarm` belief net [Beinlich et al. 1989], which has 37 nodes and 47 arcs, would require approximately 10^{17} parameters in the naïve tabular representation — *á la* Table 1. The actual belief net, however, only includes 752 parameters.[8] Essentially all of the ideas expressed in this paper apply also to related techniques for factoring a distribution; see especially the work on HMMs [Rabiner and Juang 1986] and Factorial HMMs [Ghahramani and Jordan 1997], and a description of how HMMs are related to belief nets [Smyth et al. 1997].

Of course, not every distribution can be factored. We can still represent a non-factored distribution using a belief net, albeit one that uses the comprehensive set of parameters; *e.g.*, $2^k - 1$ parameters if all k variables are binary. That is, while a belief net can exploit a factorable distribution, this formalism does not force a representation to be factored if that is inappropriate.

When we can combine this notion of probability with utility functions (which specify the "goodness" of the various possible outcomes), the resulting decision-theoretic system can often address the issues of nonmonontic inference discussed above. In particular, there is nothing problematic about deciding that `actionA` is the "optimal decision, given data η", but that `actionB` (which may "contradict" `actionA`) is appropriate given data $\eta + \Delta$. Also, while there are many ways to deal with uncertainty, etc. — including fuzzy logic, Dempster-Shafer theory of evidence, and even some forms of Neural Nets — we will focus on systems based on (standard) notions of probability [Feller 1966].

There are many obvious connections between the logic-based and probability-based formalisms. For example, an "extension" to a knowledge base KB is a complete assignment of truth or falsity to each variable, such that the conjunction of

[7] This figure includes only the $P(\chi = 1 \mid \ldots)$ entries; it omits the superfluous $P(\chi = 0 \mid \ldots)$ columns of the CPtables, as these values are always just $1 - P(\chi = 1 \mid \ldots)$.

[8] There are other tricks, such as "NoisyOr" representation, that can further reduce the number of parameters required [Pearl 1988]. For example, the well-known CPCS belief net would require 133,931,430 parameters if dealing with explicit CPtables. (Note this is still a huge savings over its unfactored table-form.) By using NoisyOr and NoisyMax's, however, this network can be represented using only 8,254 parameters [Pradhan et al. 1994].

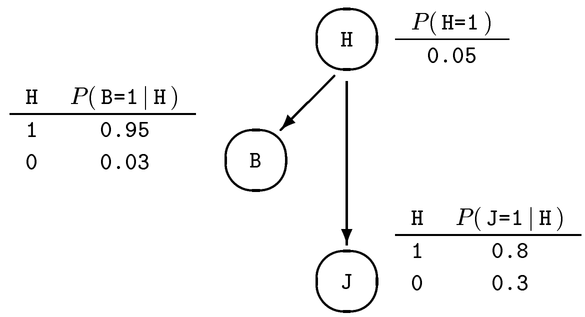


Fig. 2. Corresponding, but minimal, Belief Network

these literals entails KB. (Hence, the assignment $\{\neg A,\ B,\ C,\ \neg D\}$ is an extension of the theory $\texttt{KB}_0 = \{A \Rightarrow B,\ C \vee D\}$, as $\neg A \wedge B \wedge C \wedge \neg D \models \texttt{KB}_0$.) We can in general view the possible extensions of a knowledge base as a "qualitative" version of the atomic events (see Table 1), with the understanding that each of these "possible words" has a non-0 probability, while each of the remaining possible "complete assignments" (*e.g.*, $\{A,\ \neg B,\ \neg C,\ \neg D\}$) has probability 0 of occurring. Many, including Nilsson [Nilsson 1986], have provided formalisms that attempt to link these areas.

Note that the standard models of probability — and hence, Belief Nets — can be viewed as a natural extension of *propositional* logic, as the fundamental unit being considered is a *proposition* (*e.g.*, the particular subject is female, or is pregnant). Predicate calculus is more general than propositional logic, as its basic unit is an *individual*, which may have certain properties (*e.g.*, perhaps the individual `Mary` has the property `female`, but not the property `pregnant`). There are several groups actively trying to extend probabilities to be more expressive; see [Halpern 1990], [Ngo and Haddawy 1995], [Bacchus et al. 1996], [Koller et al. 1997; Koller and Pfeffer 1997], [Poole 1993b]. That fascinating work is beyond the scope of this survey.

For more information about belief nets, see [Pearl 1988], or
`http://www.cs.ualberta.ca/~greiner/bn.html`.

2.3 Challenges of Reasoning

A reasoning system must address (at least) the following two challenges: First, *the system must somehow contain the relevant knowledge*. There are two standard approaches to acquiring the information:

knowledge acquisition: acquire the relevant information by interviewing one or more human domain experts; see [Scott et al. 1991] for standard protocols and interviewing techniques.

learning: gather the required information from "training data" — information that typically specifies a set of situations, each coupled with the correct response [Mitchell 1997].

or, quite often, a combination of both [Webb et al. 1999].

In any case, the system builder must ensure that the language used to encode the information is sufficient; *e.g.*, if the color of an object is important (*e.g.*, to making some decision), then the language must include something like a "`colorOf(·, ·)`" predicate. Even after deciding *what* to represent (*i.e.*, the relevant ontology), the designer must also decide *how* to represent this information — *e.g.*, "`colorOf(·, red)`" vs "`red(·)`". There have been some efforts to streamline this process, and to standardize the relevant components; see the KIF project, with its advocates [Neches et al. 1991; Genesereth and Fikes 1992] and opponents [Ginsberg 1991]. There are also more fundamental encoding issues, such as the decision to represent the world using a conjunction of propositions (*i.e.*, a knowledge base), as opposed to an equivalent set (disjunction) of characteristic models; see [Kautz et al. 1993; Khardon and Roth 1994]. (This is related to the dual representations of a waveform: "time domain" vs "frequency domain" [Bracewell 1978].)

The first issue ("what to represent") is clearly crucial: if the designer produces a representation that cannot express some critical aspect of the domain, the reasoner will be unable to provide effective answers to some questions. The reprocussions of not adequately dealing with the other issue (*e.g.*, using the time domain, rather than frequency; or vice versa) are not as severe, as everything that can be expressed one way can be expressed in the other. However, the different representations may differ in terms of their "naturalness" (*i.e.*, people may find one more natural than another), and "efficiency".

In this report, however, we assume that the available information (both general and situation specific knowledge) is sufficient to reach the appropriate conclusion — although it may not be obvious how to reach (or even approximate) that conclusion; see below. As such, we will not directly consider the issues of learning domain knowledge,[9] nor will we explicitly consider the related challenges of maintaining and updating this knowledge. This survey, instead, focuses on the second major challenge: *efficiently producing all-and-only the correct answers to all relevant queries*.

2.4 Quality Measures

We would like a reasoning system that is

correct:	always returns the correct answer
precise:	always returns the most specific answer
expressive:	allows us to express any possible piece of information, and ask any possible question
efficient:	returns those answers quickly.

see [Greiner and Elkan 1991; Doyle and Patil 1991].

Unfortunately, this is impossible: first order logic is not decidable. In particular, no "sound" and "complete" (read "correct and precise") reasoning system can be decidable for a representation as expressive as arithmetic [Nagel and Newman 1958; Turing 1936]!

There are also hardness results for less expressive systems: *e.g.*, general proposi-

[9]Although we will later briefly consider learning *control* knowledge; see Section 6.1.

tional reasoning is NP-complete [Garey and Johnson 1979] (in fact, $\#P$-hard [Valiant 1979]), as is probabilistic reasoning in the context of Belief Nets [Cooper 1990]. Moreover, even getting approximate answers from a belief net, within an additive factor of 1/2, is NP-hard [Dagum and Luby 1993], as is getting answers that are within a multiplicative factor of $2^{n^{1-\epsilon}}$ for any $\epsilon > 0$ [Roth 1996].

We can view this as general property of reasoning:

> **Fundamental Tradeoff:** The worst-case run-time efficiency of any correct-and-precise reasoning process increases monotonically with the expressiveness of the reasoner's language.

Any system that wants to guarantee efficient reasoning must therefore sacrifice something — expressiveness, precision or correctness. The next three sections consider the possibility of improving the worst-case efficiency by (resp.) reducing *expressiveness*, allowing *imprecise* answers, and allowing occasional *incorrect* responses; Section 6 then considers producing (expressive, precise and correct) systems whose *"average-case"* efficiency is as high as possible. Of course, complete precision may be overkill for some tasks; *e.g.*, to decide on our next action, we may just need to know whether or not $P(\texttt{cancer}) > 0.5$; here additional precision will not be additionally useful. In general, we can identify each task with a performance criteria, then evaluate a reasoning system based on this criteria. Section 6.3 addresses this range of issues.

3. IMPROVING WORST-CASE EFFICIENCY BY <u>REDUCING EXPRESSIVENESS</u>

This section discusses reasoning systems that reduce expressiveness to obtain guarantees of efficient performance.

Less Expressive Logic-Based Reasoners: Standard "database management sytems" (DBMS) are very inexpressive, as they allow only conjunctions of positive ground atomic literals [van der Lans 1989]. These systems do allow users to state "`McD makes FrenchFries`" and "`Army makes Tanks`", and to answer questions that correspond to existentially quantified boolean combinations of such atomic literals. However, they do not allow the user to explicitly state claims of the form "`McD makes either FrenchFries or Tanks`", "`McD makes something`" nor "`McD does not make Tanks`" — that is, typical DBMS do not allow disjunctions, negations, or existentials [Reiter 1978a]. The upside is that database "reasoning" (*i.e.*, answering standard SQL queries) is efficient — at worse linear (in the size of the database).

Two comments are relevant here: First, linear efficiency may seem optimal, as it takes this much time simply to input the information. However, a clever reasoning system may be able to do better *at query (run) time* (*i.e.*, in the `Ask` routine), if it has first done some appropriate work when the information was asserted (*i.e.*, at "compile time"), via `Tell`. As an extreme, imagine a DBMS that explicitly stores the answers to all allowed queries, after all assertions have been entered; given a sufficiently narrow space of possible queries and a sufficiently good indexing scheme, a reasoner could answer queries in time considerably less than $O(n)$ — this can be extremely important in practice, as linear complexity might still be too expensive for large databases. We will later return to the general issue of *when* to perform

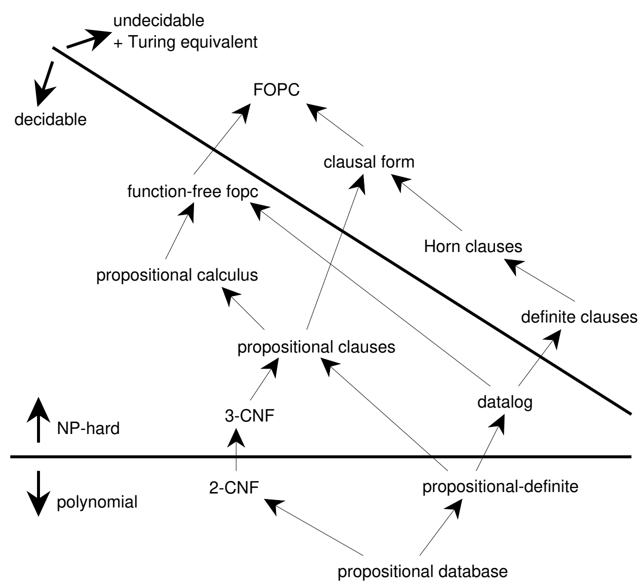


Fig. 3. Complexity Cliff (taken from Figure 5.3 of [Poole et al. 1998])

inference (Section 6.1).

Second, many database systems embody implicit assumptions that extend the set of queries that can be answered categorically. In particular, the "Closed World Assumption" allows a DBMS to conclude that "`McD does not make Tanks`" from a database that does *not* explicitly include the assertion "`McD makes Tanks`" [Reiter 1978b]; this is a special case of the "Negation As Failure" policy of many logic programs [Clark 1978]. The "Unique Names Assumption" allows a DBMS to conclude that "`McD makes (at least) two products`" from a database that includes "`McD makes Hamburgers`" and "`McD makes FrenchFries`" [Reiter 1980] as this assumption allows us to conclude that `Hamburgers`≠`FrenchFries` from the fact that their names are distinct; [10] see also [Reiter 1987]. Note that these assumptions extend the set of queries that can be answered; they do not extend the information that the user can express, as (generally) the user does not have the option of not expressing these assertions.

"Semantic Nets" and "Frame-based Systems" are alternative artificial intelligence representation formalisms. In hindsight, we can view much of the research in these areas as guided by the objective of producing an efficient, if inexpressive, reasoning system [Findler 1979]. These systems in general allows only conjunctions of atom-

[10] Note that this claim is not always true: *E.g.*, "`2+2`" and "`4`" are the same, even though they have different names; similarly both "`Professor Greiner`" and "`Russ`" refer to the same thing; as do "`MorningStar`" and "`EveningStar`" [McCarthy 1977]. Hence, if there are k entries of the form `McD makes` x_i, the Closed World Assumption allows a reasoner to conclude that McDonalds makes at most k items, and the Unique Names Assumption, that McDonalds makes at least k items.

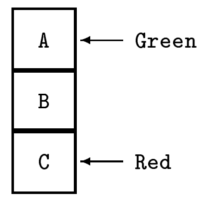


Fig. 4. Reasoning by Cases

ic unary- or binary- literals, as well as certain subclasses of simple 1-antecedent rules using such literals. The more recent work on "Terminological Logics" (aka "Description Logics") is explicitly trying to formalize, and extend, such systems, towards producing a system whose language is as expressive as possible, with the constraint that its worst time capacity must remain polynomial [Donini et al. 1997]. This relates to the notion of "complexity cliffs" [Levesque and Brachman 1985; Levesque 1984]: one can keep increasing the expressiveness of a language (by adding new "logical connectives", or increasing the number of disjuncts allowed per clause, etc.) and retain worst-case polynomial complexity, until reaching a cliff — one more extension produces a system in which inference is NP-hard. Then one can add a set of additional connectives, etc., until reaching the next cliff, where the complete system goes from decidable (if NP-hard) to undecidable. See Figure 3

As mentioned above, *resolution* is sufficient to answer any logic-based query. PROLOG is a specific embodiment of the "resolution" derivation process, which is honed to deal with a certain class of knowledge bases and queries. In particular PROLOG deals only with "Horn clauses" — *i.e.*, knowledge bases that can be expressed as conjunctions of disjunctions, where each disjunction includes at most one positive literal [Clocksin and Mellish 1981]. To motivate this restrictions, note that, while propositional reasoning is NP-hard, there is a linear time algorithm for answering queries from a Horn database [Dowling and Gallier 1984].[11] In exchange for this potentially exponential speed-up, however, there are statements that cannot be expressed in PROLOG — in particular, one cannot state arbitrary disjuncts, *e.g.*, that "`Patient7 has either Hepatitis or Meningitis`". Moreover, PROLOG cannot prove that a (existentially quantified) query is entailed unless there is a specific instantiation of the variables that is entailed. Note this is not always the case: consider a tower of 3 blocks, with the green-colored block `A` immediately above `B`, and `B` immediately above the red-colored block `C`; see Figure 4. Now observe that the answer to the question: "Is there a `green` block immediately above a non-`green` block?" is `yes`, as this holds whether `B` is `green` (and hence the `green` `B` is above the non-`green` `C`) or `B` is not `green` (and hence the `green` `A` is above the non-`green` `B`) [Moore 1982]. Fortunately, in practice, these limitations are not that severe — very few standard tasks require such "reasoning by cases" and implicitly-specificed answers.[12]

[11] Moreover, PROLOG uses a type of "ordered resolution"; ordered resolution is refutation complete for Horn clauses [Genesereth and Nilsson 1987]. Also, [Boros et al. 1990] and [Dalal and Etherington 1992b] provide yet other syntactic situations for which reasoning is guaranteed to be efficient.

[12] Also, as PROLOG does not perform an "occurscheck", it is not sound — *i.e.*, it can return

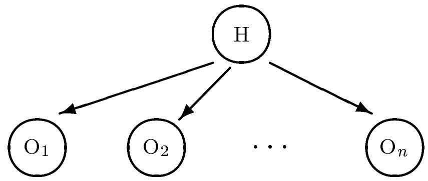


Fig. 5. Example of "Naïve Bayes" Structure

There are also constaint programming systems [Freuder 1996] that attempt to accomodate a larger subset of clauses — of course, such systems cannot provide the efficiency guarantees that a Horn-clause system can.

Less Expressive Belief Nets, Probabilistic Queries: We now consider the work on probabilities and belief nets, focusing on the "belief updating" task: *i.e.*, computing $P(H = h \mid E_1 = e_1, \ldots, E_m = e_m)$, the posterior probability that the hypothesis variable H has value h, conditioned on some concrete evidence, of the form `Cough = true`, `Temp = 100`^{o}, ...[13]

We note first that there are efficient inference processes associated with certain types of structures, and for certain types of queries. Inference is trivial for "naïve-bayes" classifiers, of the form shown in Figure 5. These $n+1$-node belief nets were motivated by the task of classification: *i.e.*, assigning a "classification label" to an instance, specified by a set of (up to n) attribute values. Each such net includes one node to represent the "classification", which is the (only) parent of all of the other nodes (the "attributes"). While inference is guaranteed to be fast $O(r)$ (where $r \leq n$ is the number of specified attributes; *e.g.*, $P(H = 1 \mid O_1 = o_1, \ldots, O_r = o_r))$, these systems cannot express any general dependencies between the attributes, as its structure forces $P(O = i \mid O_j, H) = P(O = i \mid H)$ for all $i \neq j$.

There are also efficient algorithms for inference in the more general class of "poly trees" — *i.e.*, belief nets that include at most a single (undirected) path connecting any pair of nodes [Pearl 1988]; see Appendix A.1. Notice this class strictly generalizes tree structures (and *a fortiori*, naïve-bayes structures) by ignoring the directions of the arrows, and by allowing more complex structures (*e.g.*, allowing multiple root nodes, longer paths, etc.). However, these are still many dependencies that cannot be expressed.

Friedman, Geiger and Goldschmidt [Friedman et al. 1997] combine the ideas of naïve-bayes and poly-tree structures, to produce "Tree Augmented Bayesian net" or TAN, structures, that are typically used to classify unlabled instances. These structures resemble naïve-bayes trees, but allow certain dependencies between the children (read "attributes"). To define these TAN structures: (1) There is a link from the single classification node down to every other "attribute" node. (2) Let

---

an answer that is not correct. This too is for efficiency, as it means the unification procedure, running in the innermost loop, is $O(k)$, rather than $O(k^2)$, where k is the size of the largest literal [Genesereth and Nilsson 1987]. Prolog also includes some "impurities", such as negation-as-failure "`not(·)`" and cut "`!`".

[13] Many other standard tasks — such as computing the *maximum a posteriori* assignment to the variables given some concrete evidence — require similar algorithms, and have similar computational complexity [Dechter 1998; Abdelbar and Hedetniemi 1998].

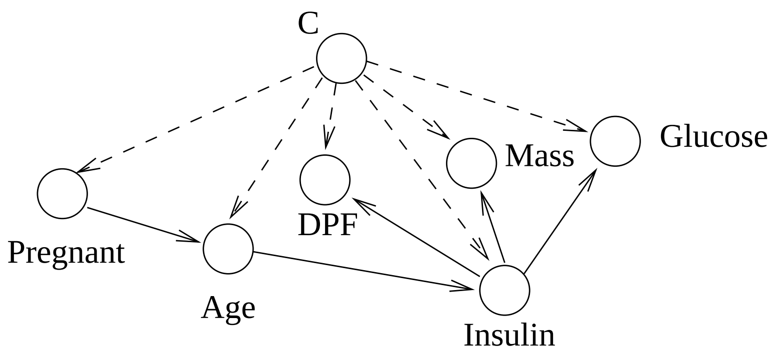


Fig. 6. A TAN model: dashed lines are from naïve bayes; solid lines express correlation between attributes (taken from Figure 3 of [Friedman et al. 1997])

BN' be the structure obtained by removing these classification-to-attribute links from the initial TANetwork. This BN' then is a poly-tree. See Figure 6.[14]

We close our comments about efficient *structures* with a few quick comments: (1) All three of these classes (naïve bayes, poly-tree, and TANetworks) can be learned efficiently [Chow and Lui 1968], as well as admiting efficient reasoning. (This is a coincidence, as there are classes of networks that admit efficient inference but are hard to learn, and vice versa.) (2) Many system designers, as well as learning algorithms [Singh 1998; Sarkar and Murthy 1996] use the heuristic that "networks with fewer arcs tend to be more efficient", as an argument for seeking networks with fewer connections.[15] Of course, belief net inference can be NP-complete even if no node in the network has more than 3 parents [Cooper 1990]. (3) There are also efficient algorithms for reasoning from some other simple structures, such as "similarity networks" [Heckerman 1991].

These positive results deal with general queries, where only a subset of the possible conditioning variables have been set. There are also efficient ways to compute $P(\,H \mid E_1 = e_1, \ldots, E_m = e_m\,)$ from *any belief net*, provided the evidence set $\{E_i\}$ is comprehensive — *i.e.*, includes all variables, or all variables other than H. Actually, it is easy to compute $P(\,H \mid E_1 = e_1, \ldots, E_m = e_m\,)$, from a general belief net, if the evidence $\{E_i\}$ includes of H's parents and none of H's descendents; or if $\{E_i\}$ includes H's "Markov blanket": that is, all of H's parents, H's children and H's "co-parents" (all of the non-H immediate parents of all of H's immediate children — *e.g.*, refering to Figure 6, `C` and `Pregnant` are co-parents of `Age`; and `C` and `Age` are co-parents of `Insulin`).

By contrast, the most efficient known algorithms for answering general queries from *general* belief nets are exponential. The algorithms based on "junction tree"

[14] To see why there is an efficient inference algorithm, just observe that a TAN structure has a single node cut-set; see Appendix A.2.2.
[15] Note this point is orthogonal to the goal of learning more accurate networks by using a regularizing term to avoid overfitting [Heckerman 1995].

(aka clique-tree — *cf.*, bucket elimination [Dechter 1998]) are exponential in the network's "induced tree width" — a measure of the topology of the network [Arnborg 1985]. Another class of algorithms is based on "cut set elimination". Here, the complexity is exponential in the "min cut" (for d-separation) of the network. Appendix A.2 provides a quick summary of the major algorithms for computing posterior probabilities using a belief net.

Summary: This section has considered several types of reasoners; each always produces correct and precise answers, *over the situations it can accomodate*: that is, *if* you can state the appropriate facts, and pose the appropriate question, then the system will produce all-and-only the correct answers, efficiently. There are, however, limitations on what can be stated and/or asked.

The next sections present other reasoners that attempt to remain as expressive as possible, but hope to gain efficiency by being imprecise or occasionally incorrect, etc.

4. IMPROVING WORST-CASE EFFICIENCY BY ALLOWING VAGUE ANSWERS

In general, a reasoning algorithm produces an answer to each given query. This answer is *correct* if it follows from the given knowledge base. Note that a correct answer can still be **vague** or **imprecise**. For example, given $P(\texttt{Hep}\,|\,\texttt{Jaundice}) = 0.032$, the answer "$P(\texttt{Hep}\,|\,\texttt{Jaundice}) \in [0, 0.1]$" is correct, but less precise. At the extreme, the answer "$P(\texttt{Hep}\,|\,\texttt{Jaundice}) \in [0, 1]$", while so vague as to be useless, is not wrong. Similarly, answering a propositional query with "IDK" (for "I don't know") is not incorrect; this vague answer is, in many situations, better than arbitrarily guessing (say) "No". As a less extreme situation, a system may that answers the question "`What is the disease?`" with "`a bacteria`" is correct, but less precise than stating "`enterobactericaiea`"; similarly a correct (if vague) answer to "`Who are the Stan's uncles?`" could be "`7 adult men, all living in Kansas`"; or perhaps "`at least 2 adult men, and at least one living outside California`"; etc.

Precision, like correctness, is relative to a given knowledge base. For example, stating that "`Fido IsA Dog`" is precise if that is all that the KB sanctions. However, this answer is imprecise with respect to a more specific KB that entails "`Fido IsA Chihuahua`". This is especially true for systems that deal with "qualitative reasoning" [Weld and de Kleer 1990] or "qualitative belief nets" [Wellman 1990]. In general, we say an answer is *precise (with respect to a knowledge base KB and query)* if it is as specific as possible; and otherwise is considered **imprecise** (or vague, approximate). Note that this issue of precision is orthogonal to correctness — an answer can be precise, but wrong; *e.g.*, imagine stating $P(\texttt{Hep}\,|\,\texttt{Jaundice}) = 0.8$ when it is actually 0.7. Of course, in some situations (correct but) vague answers may be sufficient — *e.g.*, you may only need to know whether $P(\texttt{StockX goes up}\,|\,\cdots) > 0.5$ to decide on your next action; Section 6.3 explores this topic. In this section, we will consider reasoners that always return correct, but perhaps vague, answers.

To state this more precisely: In general, let α be an answer to the query φ; in the predicate calculus situation, this may correspond to $\varphi[\beta]$ for some binding β, or perhaps $\varphi[\beta_1]\&\cdots\&\varphi[\beta_n]$ if the user was seeking all answers. In the probabilistic

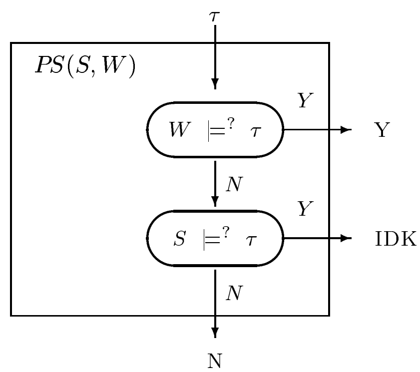


Fig. 7. Flow Diagram of $PS(S,W)$ addressing $\Sigma \models^? \tau$

case, where φ is $P(\,A \,|\, B\,)$, α could be the "$P(\,A \,|\, B\,) = 0.4$" or "$P(\,A \,|\, B\,) \in [0, 0.5]$". Assume the correct, precise answer for the query φ is φ^*; this means know that $KB \models \varphi^*$. The answer α is correct, but not necessarily precise, if $\models \varphi^* \Rightarrow \alpha$ (where α must also satisfy some syntactic requirements, to insure that it relates to the φ query).[16] We view IDK as the tautology "`true or false`"; notice this answer is always correct. We can also consider a relative measure of precision, saying that α_1 is more precise than α_2 if $\models \alpha_2 \Rightarrow \alpha_1$; in fact, this induces a partial ordering. Note finally that a correct answer α is considered precise if also $\models \alpha \Rightarrow \varphi^*$.

Imprecise Logical Reasoners: Many imprecise reasoners work by removing some (hopefully inconsequential) distinction from the knowledge base and the query, in order to simplify the reasoning process. This has lead to the large body of work on *approximation*, often coupled with the related notion of abstraction [AAAI 1992; Ellman 1993; Sacerdoti 1973].

One specific example is a "Horn Approximation" reasoner [Selman and Kautz 1996]. These systems are motivated by the observation that, while inference from a general propositional theory is NP-hard, there are efficient algorithms for reasoning from a Horn theory. Now note that we can always *bound* a non-Horn theory Σ by a pair of Horn theories $\langle\, S,\, W\,\rangle$. (*E.g.*, given $\Sigma = \{a \vee b,\ \neg b \vee c\}$, $S = \{a, \neg b \vee c\}$ is a "stronger (Horn) theory" while $W = \{\neg b \vee c\}$ is a "weaker (Horn) theory".) The reasoner, called $PS(S,W)$ in Figure 7, can use this pair $\langle\, S, W\,\rangle$ — called a "Horn approximation to Σ" — to answer $\Sigma \models^? \tau$: It first asks whether $W \models^? \tau$, and returns "Yes" if that query succeeds. Otherwise, it asks whether $S \models^? \tau$, and returns "No" if that query fails. If neither test produces the definitive answer (*i.e.*, if $W \not\models \tau$ and $S \models \tau$), the reasoner simply returns the imprecise "IDK"; see Figure 7.[17] Note that the answers are always correct (*e.g.*, $W \models \tau$ implies $\Sigma \models \tau$ and $S \not\models \tau$ implies $\Sigma \not\models \tau$); but they are not always as precise as the answers that would arise from the original Σ. (That is, if $W \not\models \tau$ and $S \models \tau$,

[16]Technically, we should write $KB^- \models \varphi^* \Rightarrow \alpha$ where KB^- contains the information required to connect the notation in φ^* to α; *e.g.*, to prove that $A = 0.3$ implies $A \in [0, 0.4]$.
[17]Dalal and Etherington [Dalal and Etherington 1992a] discuss various extensions to this framework.

then $\Sigma \models \tau$ would return a categorical answer, but $PS(S, W)$ does not.) There are many challenges here — *e.g.*, there are many "maximal" strengthenings (which are NP-hard to find), and the optimal weakening can be exponentially larger than the initial theory. Section 6 below discusses an approach to address these issues.

We can also consider stochastic algorithms here. One recent, prominent example is the GSAT algorithm [Selman et al. 1992], which attempts to solve satisfiability problems by hill-climbing and plateau-walking in the space of assignments, starting from a random initial assignment. That is, the score of an assignment, for a fixed SAT formula with m clauses[18], is the number of clauses that are satisfied; note this score ranges from 0 to m. At each stage, given a current assignment, GSAT sequentially considers changing each individual variable of that assignment. It then "climbs" to the new assignment with the highest score, and recurs. Here, if GSAT finds an satisfying assignment (*i.e.*, an assignment with the score of m), it correctly reports that the problem has a solution. As GSAT is unable to determine if the SAT instance has no solution, it will terminate after some number of iterations and random walks and return the inconclusive "I don't know". As such, GSAT is a "LasVegas algorithm" [Hoos and Stutzle 1998], which knows when it knows the answer, and so only returns the correct answer, or is silent.

It is worth commenting that, despite its inherent incompleteness, GSAT has proven extremely useful — being able to solve problems that no other current algorithm can solve. It, and related ideas, have since been used, very successfully, in planning research [Kautz and Selman 1996].

Imprecise Probabilistic Reasoners: As mentioned above, a probabilistic reasoner that returns only $P(\texttt{Hep} \mid \texttt{Jaundice}) \in [0, 0.10]$, when the correct answer is $P(\texttt{Hep} \mid \texttt{Jaundice}) = 0.032$, is correct, but imprecise. Unfortunately, for a given belief net, even finding approximately correct answer is hard; *e.g.*, as noted above, getting an answer within an additive factor of $1/2$, is NP-hard [Dagum and Luby 1993], as is getting answers that are within a multiplicative factor of $2^{n^{1-\epsilon}}$ for any $\epsilon > 0$ [Roth 1996].

There are, however, many systems that address this approximation task. Here we include algorithms that return answers that are guaranteed to be under-bounds, or over-bounds, of the correct value, as we view an underbound of p as meaning the answer is guanteed to be in the interval $[p, 1]$; and an overbound of q means it is in $[0, q]$.

Many approximation techniques are best described relative to the algorithm used to perform exact probabilistic inference, in general. (1) In [Dechter and Rish 1997], Dechter and Rish modify their Bucket Elimination algorithm (see Appendix A.2.1) to provide an approximate answer; they replace each bucket's function with a set of "smaller" functions that each include only a subset of the functions associated with each variable. Those results hold for discrete variables. Jaakkola and Jordan [Jaakkola and Jordan 1996a; Jaakkola and Jordan 1996b] use a similar idea for continuous (Gaussian) variables: their algorithm sequentially removes each unassigned variable, replacing it with an approximation function that computes an

[18] Each such formula is a specific conjunction of m clauses, where each clause is a disjunction of literals, where each literal is either a boolean variable or its negation [Garey and Johnson 1979].

under-bound (resp., over-bound) to the correct function.

(2) Horvitz and others have similarly "tweaked" the idea of context of cut-set conditioning (see Appendix A.2.2): Rather than consider all values of the cut-set variables, they get an approximation that is guaranteed to be within an additive factor of ϵ away from the correct value by summing over only those values whose probability mass collectively exceeds $1-\epsilon$; see [Horvitz et al. 1989].

Other approaches work by removing some aspect of the belief net — either node, or arc, or values of a variable. The LPE (Localized Partial Evaluation) algorithm [Draper and Hanks 1994] maintains and propagates *intervals* during the inference process, and so can compute a range for the posterior probability of interest. LPE can disregard a variable (perhaps because it appears distant from the query nodes, or seems relatively irrelevant) by setting its value to $[0, 1]$. (In practice, this means the inference algorithm has fewer nodes to consider, and so typically is faster.) See also the work by Srinivas [Srinivas 1994] and Mengshoel and Wilkins [Mengshoel and Wilkins 1997], who also consider removing some nodes from networks, to reduce the complexity of the computation.

Others attempt to reduce complexity by removing some *arcs* from a network. In particular, Kjaerulff [Kjærulff 1994] proposes some heuristics that suggest which arcs should be removed, with respect to the clique-tree algorithm; see Appendix A.2.1. Van Engelen [van Engelen 1997] extends this idea by providing an algorithm that applies to arbitrary inference algorithms (not just the clique-tree approach) and also allows, in some cases, many arcs to be removed at once. He also bounds the error $|P_{N_r \not\to N_s}(\,H \mid E\,) - P(\,H \mid E\,)|$ obtained by using $\mathcal{B}_{N_r \not\to N_s}$ rather than (the correct) $\mathcal{B}$ to compute this conditional proability. (Here, $\mathcal{B}_{N_r \not\to N_s}$ is the network obtained by removing the $N_r \to N_s$ arc from the network $\mathcal{B}$, $P_{N_r \not\to N_s}(\,H \mid E\,)$ is the probability value returned by this network, and $P(\,H \mid E\,)$ is the result obtained by the original network.)

The Wellman and Liu approximation algorithm [Liu and Wellman 1997; Wellman 1994] leaves the structure of the belief net unaffected; it reduces the complexity of inference by instead abstracting the state of some individual variables — *e.g.*, changing a variable that ranges over 10 states to one that only ranges over (say) 4 values, by partitioning the set of values of the original variable into a smaller set of subsets.

A related approach reduces the complexity of inference by disregarding values of variables which are relatively unlikely [Poole 1993a]. This reduction is especially large when the relevant variable(s) occur early in a topological ordering used by the reasoner; *e.g.*, when each variable occurs before all of its children.

Recently, Dagum and Luby [Dagum and Luby 1997] proved that networks without extreme values can be approximated efficiently. They begin with the observation that the hardness proofs for approximation tasks all involve CPtables that include 0s and 1s — which, in essence, means the network is being used to do something like logical inference. In general, if all of the values of the CPtables are bounded away from 0 (*i.e.*, if all entries are greater than some $\lambda > 0$), then there is a sub-exponential time algorithm for computing a value that is guaranteed to be within a (bounded) multiplicative factor of optimal — *i.e.*, an algorithm that returns an answer $\hat{v}$ such that $(1-\epsilon)\,v \;\leq\; \hat{v} \;\leq\; (1+\epsilon)\,v$ where v is the correct answer and ϵ is a parameter supplied by the user, in time $O(\text{poly}(1/\epsilon)\, 2^{(\ln n)^{O(d)}})$ where d is the "depth" of belief net and n is number of nodes.

Comments: There are two obvious ways an approximate reasoner can interact with the user: First, the user can set some precision bounds, and expect the reasoner to return an answer that satisfies this requirement — although perhaps the reasoner will "charge" the user for this, by requiring more time to return an answer that is more precise. Of the results shown above, only Dagum/Luby has this property. The others, instead, all return both an answer and the bounds, where the bounds specify how precise that answer is. If the resulting answer/bounds are not sufficient, the user can often adjust some parameters, to get another answer/bound pair, where the precision (read "bound") is presumably better.

Note that all of these systems improve the efficiency of the underlying computation, but produce an answer that is an only an approximation to the correct value. There are also a body of tricks that can sometimes reduce the *average* complexity of answering a query, but at no degradation of precision; we mention some such tricks in Section 6.

5. IMPROVING WORST-CASE EFFICIENCY BY ALLOWING INCORRECT RESPONSES

Of course, not all reasoners provide answers that are always correct; some may occasionally return erroneous answers. We can evaluate such not-always-correct reasoners R using a distance measure $d(a_q,\, R(q)) \;\in \Re^{\geq 0}$ that computes the distance between the correct answer a_q to the query q and the response that R returned $R(q)$. (Note this $d(a, b)$ function could be 1 iff $a \neq b$, or 0 otherwise; such a function just determines whether R returned the correct answer or not.) R's "worst-case" score will be largest such value over all queries q, $\max_q\{d(a_q,\, R(q))\}$; and its "average score" is the expected value, $E_q[\, d(a_q, R(q))\,] \;=\; \sum_q P(\text{ "}q\text{" asked}) \times d(\, a_q, R(q)\,)$, over the queries encountered. To evaluate a stochastic reasoner, we also average over the answers returned for any setting of the random bits.

We will sub-divide these not-correct (aka "unsound") systems into two groups: those which are **deterministic** (each time a query is posed, it returns the same possibly-incorrect answer) versus those which are **stochastic** — and may return *different* answers to *same query*, on different calls.

As one example of a well-motivated deterministic unsound algorithms, consider anytime algorithms [Zilberstein 1993; Zilberstein 1996]: While addressing a specific problem, these systems are required to produce an answer whenever requested, even before the system has run to completion. (Note this is appropriate if the reasoner really has to return an answer, to decide on some action before a hard deadline. Of course, "no action" is also a decision.) Here, these systems may have to guess, if they have not yet computed the appropriate answer. As another example, consider the variant of Horn approximation (see Figure 7) that returns `Yes`, rather than IDK, if the definitive tests both fail.

We will, however, focus on stochastic algorithms, where the reasoner uses some source of random bits during its computation.

Stochastic Logical-Reasoning Algorithms: As one obvious example, we can motify GSAT to return "NOT-satisfiable" (rather than "IDK") if no satisfying assignment is found. Once again, this is not necessarily correct. If this algorithm

```
LogicSample(BN, X, E⃗ = e⃗ )
        % Returns estimate P̂(X = x_i | E⃗ = e⃗) of P( X | E⃗ = e⃗ ), from belief net BN
  Let M := 0
  For all x_i in Dom(X)   Let m_i := 0
  Do until done:        % based on # of samples, or some threshold, or ...
    r = GetRandom(BN)
    If r[E⃗] = e⃗,   then M += 1
    If r[X] = x_i and r[E⃗] = e⃗,   then m_i += 1
  Return  ⟨ m_i/M ⟩_i     % as estimates of P̂(X = x_i | E⃗ = e⃗) for each x_i ∈ Dom(X)
```

```
GetRandom(BN)
        % Returns random instance r⃗ from BN, which maps each node in 𝒩 to
        % a value in its domain
  For each root node T ∈ 𝒩
    Draw t from Dom(T) with prob P_BN( T = t_i )
    Set  r⃗[T] := t_i
  For each remaining node W (in order satisfying DAG-structure)
    After finding values for all W's parents   U_1 = u_1, ..., U_k = u_k,
      Draw w_j from Dom(W) with prob P_BN( W = w_j | U⃗ = u⃗ )
      Set  r⃗[W] := w_j
  Return  r⃗
```

Fig. 8. Logical Sampling

has performed a sufficient number of restarts, however, we can still be confident that "NOT-satisfiable" is correct.

Stochastic Probabilistic-Reasoning Algorithms: There is a huge inventory of algorithms that stochastically evaluate a belief net. The simplest such algorithm, "Logic Sampling", appears in Figure 8. This algorithm simply draws a number of samples (from the distribution associated with the belief net), and computes the empirical estimate of the query variable: of the times that the conditioning event $\vec{E} = \vec{e}$ occured (over the instances generated), how often was X equal to x_i, for each x_i is X's domain.

This algorithm uses GetRandom(BN). To explain how this subroutine works, consider the BN_{WG} network shown in Figure 9. Here, GetRandom(BN_{WG}) would ...

(1) Get a value for "`Cloudy`", by *Flipping* 0.5-*coin*
  Assume flip returns "`Cloudy = True`"
(2) Get a value for "`Sprinkler`", by *Flipping* 0.1-*coin*
  (as `Cloudy = True`, $P(\,\mathtt{S}\,|\,\mathtt{C{=}T}\,) = 0.1$)
  Assume flip returns "`Sprinker = False`"
(3) Get a value for "`Rain`", by *Flipping* 0.8-*coin*
  (as `Cloudy = True`, $P(\,\mathtt{R}\,|\,\mathtt{C{=}T}\,) = 0.8$)
  Assume flip returns "`Rain = True`"
(4) Get a value for "`WetGrass`", by *Flipping* 0.9-*coin*
  (as `Sprinkler = F`, `Rain = T`, $P(\,\mathtt{W}\,|\,\neg\mathtt{S},\ \mathtt{R}\,) = 0.9$)

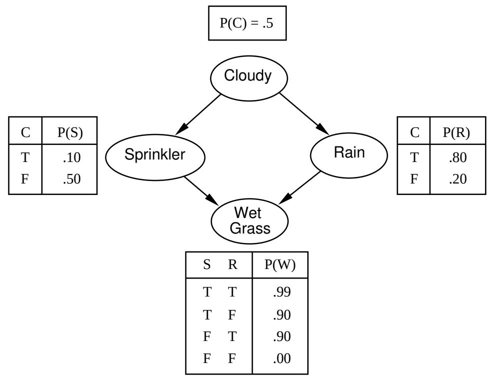


Fig. 9. Simple Belief Net BN_{WG} (taken from Figure 15.9 of [Russell and Norvig, 1995])

Assume flip returns "`WetGrass = True`"

Here, GETRANDOM(BN_{WG}) would return the instance

⟨`Cloudy = True`, `Sprinker = False`, `Rain = True`, `WetGrass = True`⟩.

On other calls, GETRANDOM(BN_{WG}) would see different results of the coin-flips, and so return other instances.

Note that the LOGICSAMPLE algorithm will, in essense, ignore an instance $\vec{r}$ if $\vec{r}[\vec{E}] \neq \vec{e}$. (Here $\vec{r}[\vec{E}]$ corresponds to the tuple of the values of the variables in $\vec{E}$). This is clearly inefficient, especially if $\vec{E} = \vec{e}$ is a rare event.

The LWSAMPLE algorithm (for Likelihood Weighted Sampling), shown in Figure 10, avoids this problem: The routine it uses to generate samples GETLWRANDOM($\cdot$) is similar to GETRANDOM($\cdot$), except that it insists that each generated instance $\vec{r}$ has $\vec{r}[E_i] = e_i$ for each evidence variable E_i. However, while GETRANDOM(BN_{WG}) gave each instance a score of 1 (*i.e.*, observe the `M += 1` and $\mathtt{m}_i$ `+= 1` in Figure 8), the new GETLWRANDOM(BN, $\vec{E} = \vec{e}$) routine will instead use a "weight" of $p = P(\vec{E}_i = \vec{e}_i \,|\, \vec{U}_i = \vec{u}_i)$, where $\vec{U}_i$ are $\vec{E}_i$'s parents, and $\vec{u}_i$ is the current assigment to $\vec{U}_i$.

For example, to estimate $P($`WetGrass`$\,|\,$`Rain`$)$, from BN_{WG} (Figure 9), the associated GETLWSAMPLE would...

(1) Get a value for "`Cloudy`", by *Flipping* 0.5-*coin*
Assume flip returned "`Cloudy = False`"
(2) Get a value for "`Sprinkler`", by *Flipping* 0.5-*coin*
(as `Cloudy = False`, $P($`S`$\,|\,$`C=T`$) = 0.5$)
Assume flip returned "`Sprinker = True`"
(3) Now for "`Rain`"
Note that this is an evidence variable; so set it to the required value, `True`.
As `Cloudy = False`, $P($`R`$\,|\,$`C=F`$) = 0.2$
So this "run" counts as $p = 0.2$
(4) Get a value for "`WetGrass`", by *Flipping* 0.99-*coin*
(as `Sprinkler = T`, `Rain = T`, $P($`W`$\,|\,$`S, R`$) = 0.99$)
Assume flip returned "`WetGrass = True`"

LWSAMPLE(BN, X, $\vec{E} = \vec{e}$)
  Let M := 0
  For all x_i in $Dom(X)$ Let $\mathtt{m}_i$:= 0
  Do until done: *% based on # of samples, or some threshold, or ...*
    $\langle \vec{r}, p \rangle$:= GETLWSAMPLE(BN, $\vec{E} = \vec{e}$)
      % $\vec{r}$ is a "random" instance from BN s.t. $\vec{r}[\vec{E}] = \vec{e}$
      % p is $\prod_{E_i} P(E_i = e_i \mid \vec{U} = \vec{r}[\vec{u}])$
        % where $\vec{U}_i$ are E_i's parents, and $\vec{r}[\vec{u}_i]$ is $\vec{r}$'s assignment to $\vec{U}_i$.
    M += p
    If $X = x_i$, then $\mathtt{m}_i$ += p
  Return $\langle m_i / M \rangle_i$ *% as estimates of $\hat{P}(X = x_i \mid \vec{E} = \vec{e})$ for each $x_i \in Dom(X)$*

Fig. 10. Likelihood Weighted Sampling

Here, the associated tuple is

$\langle$ `Cloudy = False`, `Sprinker = True`, `Rain = True`, `WetGrass = True` $\rangle$

and the probability is $p = 0.2$; LWSAMPLE would therefore increment both M and $m_{WG=T}$ by 0.2.

There are many other stochastic algorithms that have been used for belief net inference — *e.g.*, based on Importance Sampling, Monte Carlo sampling [Pradhan and Dagum 1996], etc. — as well as algorithms that combine stochastic algorithms for some parts, with exact reasoning for other nodes. Many of these are described in [Cousins et al. 1993].

Note, however, that adding "randomness" does not seem to help, in the worst-case, as it is probably hard to find answers that are, with high probability, approximate correct: no polynomial-time algorithm can generate randomized approximations of the probability $P(X = x \mid E = e)$ with absolute error $\epsilon < 1/2$ and failure probability $\delta < 1/2$, unless $NP \subset RP$ [Dagum and Luby 1993].

6. IMPROVING AVERAGE EFFICIENCY

Note that the specific hardness results mentioned above are "worst case" — *i.e.*, there is a parameterized class of decision problems that exhibits this exponential scaling [Garey and Johnson 1979]. What if these particular problematic problems are not actually encountered in practice? In general, we would like to implement a system whose *average efficiency* is as high as possible, where "average" is with respect to the *distribution of tasks actually encountered*. The first two subsections therefore consider ways to improve a (logic-based, probabilistic) reasoner's average efficiency. Section 6.3 extends these ideas to consider ways to improve the average *performance*, where "performance" is a user-defined function that can inter-relate correctness, precision and efficiency, each weighted as appropriate to the task.

The approaches described below relate to the ideas discussed in Section 3, of simply defining away such problematic tasks — *i.e.*, using reasoners that can only handle a subclass of tasks that excludes these expensive situations. The difference is in how the decision (of which reasoner to use) was reached. There, the decision was based (implicitly) on the hope that the problematic situations will not occur;

here instead the system knows something about which situations will occur, and so employs algorithms designed to do well for these situations.

If the system *designer* knows a lot about the way the eventual reasoner will be used, that designer could engineer in the specific algorithm that works best for this subspace of tasks. For example, if the designer knows that the attribute values are conditionally independent (given the classification), it makes sense to build a naïve-bayes classifier; similarly, if the system never needs to reason by cases, the PROLOG inference process is quite appropriate. Alternatively, the reasoner could "learn" information about this distribution of tasks, enough to identify a reasoner (perhaps from a set of parameterized reasoning algorithms) that performs best; see [Greiner 1996]. Below we will consider both situations — where the designer does, versus does not, know the performance task.

6.1 Efficient (on Average) Logical Reasoners

In general, a logic-reasoning system must decide, at each step, which inference or rewrite rule to use, and on which set of statements. Even resolution-based systems, which consider only a single inference rule (and so have no choice in that respect), must still decide which clauses to select. While general resolution schemes (such as "set of support" or "lock" [Genesereth and Nilsson 1987]) may constrain *this* selection, there can still be many possible options. The logical-reasoning community has therefore investigated a variety of techniques for improving the performance of logic-reasoning systems, often by providing some explicit meta-level control information for controlling the inference process; *cf.*, [Smith and Genesereth 1985; Smith et al. 1986]. This is clearly related to the work by the database community on join-ordering, magic sets, and the like [Swami and Gupta 1988; Azevedo 1997], and the large body of heuristics used to solve "Constraint Satisfaction Problems" [Kondrak and van Beek 1997].

Each of these optimizations is specific to a single problem instance, and many can be viewed as a preprocessing step, called before beginning to solve the problem. Note, however, that these modifications do not affect the underlying reasoning system; and in particular, if that reasoner encountered the exact same problem a second time, it would presumably follow the same (preprocess and) process steps.

As this pre-processing step is typically very expensive, another body of work performs only a single "pre-processing" step prior to solving a *set* of queries (rather than one such step before each query). These systems can be viewed as modifying the underlying reasoner, to produce a new reasoner that does well (on average) over the distribution of queries it will encounter. Note this resulting reasoner will then solve each single query directly, without first performing any additional query-specific pre-processing step.

To make this more concrete, consider a resolution system that uses a *clause ordering* Θ to specify when to use which clause during the proof process: here, it attempts to resolve the current subgoal (determined by the depth-first order) with the (untried) clause at the highest position in Θ. (This can correspond to PROLOG's control strategy, when the ordering is determined by the chronological order of assertion.) Of course, the time required to find a solution to a specific query depends critically on the clause-ordering Θ used. A useful modification, therefore, would therefore change the clause ordering to one whose average time, over the *distribution of queries*, is minimum. The resulting reasoner would then use this (new) fixed order in answering any query.

In general, the system designer may not know, at "compile time", what queries will be posed. However, given that the reasoner will be used many times, a learner[19] may be able to estimate this distribution of tasks by watching the (initial) reasoner as it is solving these tasks. The learner can then use this distributional information to identify the algorithm that will work best. Here, the overall system (including both reasoner and learner) would have to pay the expense of solving problems using standard general methods for the first few tasks, before obtaining the information required to identify (and then use) the best special purpose algorithm.

This has lead to the body of work on "explanation-based learning" (EBL): after solving a set of queries, an EBL system will analyse these results, seeking ways to solve these same queries (and related ones) more efficiently when they are next encountered; see [Mitchell et al. 1986; Greiner 1999]. As the underlying optimization task is NP-hard [Greiner 1991], most of these systems hill-climb, from an initial strategy to a better one [Greiner 1996; Gratch and Dejong 1996].

There are other tricks for improving a reasoner's average efficiency. Recall that our interface to the reasoner is via the `Tell` and `Ask` subroutines. In PROLOG, the `Tell` routine is trivial; the `Ask` routine must do essentially all of the reasoning, to answer the query. Other systems (including those built on OPS [Brownston et al. 1986], including XCON [Barker et al. 1989]) do the bulk of the inference in a "forward chaining" manner — here, the `Tell` will forward-chain to assert various newly-entailed propositions. Answering certain questions is trivial in this situation, especially if we know that the answer to the query will be explicitly present iff it is entailed. Treitel and Genesereth [Treitel and Genesereth 1987] consider "mixed" systems, where both `Ask` and `Tell` share the load: When their system is `Tell`ed a new fact, it will forward-chain, but only using certain rules and to a certain depth, to establish a "boundary" in the associated "inference graph".[20] Afterwards, when the `Ask` process is answering some query, it will backward chain, but only following certain rules, and only until just reaching the "boundary" set by the earlier forward-chaining steps. In general, a "scheme" specifies which rules are used in a forward-chaining, vs backward-chaining, fashion, etc. Finding the optimal scheme, which minimizes the total computational time (of both forward-chaining and asserting, and backward chaining and subgoaling), is challenging — as it requires knowing (or at least estimating) the distribution of what queries will be asked, as well as the distribution of the new information that will be asserted; and then (typically) solving an NP-hard problem.

This is similar to the work on caching solutions found during a derivation, to improve the performance on later queries. Chaudri and Greiner [Chaudhri and Greiner 1992] present an efficient algorithm, for a certain class of rule-sets, that specifies which results should be stored, as a function of the frequency of queries and updates, the costs of storage, etc.

[19]Note this learner is learning about the usage patterns of the reasoner, to help in providing control information, and not about the domain *per se*.

[20]This hyper-graph has a node for each proposition, connected by hyperlinks, each of which connects the conclusion of a rule to the rule's set of antecedents.

6.2 Efficient (on Average) Belief Net Inference

Most BN inference algorithms perform their inference based only on the specific belief net involved. Some systems will also examine the specific query in a "preprocessing step", and modify their algorithms and datastructure accordingly — *e.g.*, remove nodes "d-separated" from the query and evidence nodes [Pearl 1988] (including "barren nodes" [Zhang and Poole 1994]), find an ordering for the process that is appropriate given the nodes that will be instantiated [Dechter 1998] and so forth

As mentioned in the earlier logic-based situation, such preprocessing can be expensive — indeed, it could involve solving the underlying inference process, or some other NP-hard problem. An effective inference process may therefore, instead, try to find an inference procedure that is optimal *for the distribution of queries* it will encounter. Here, it will do some modification once, rather than once per query.

Herskovits and Cooper [Herskovits and Cooper 1991] consider caching the answers to the most common queries; here, the reasoner can simply return these explicitly-stored answers when those particular queries are posed; otherwise, the reasoner will do a standard belief net computation. (Those researchers used an analytic model, obtained from the belief net itself, to induce a distribution over queries, and used this to determine which queries will be most common.)

The QueryDAG work [Darwiche and Provan 1996; Darwiche and Provan 1997] takes a different cut: after the designer has specified the belief net queries that the performance system will have to handle, the QueryDag system "compiles" the appropriate inference system, to produce an efficient, well-honed system, that is efficient for these queries. (Even if this compiled version performed the same basic computations that the basic inference algorithm would perform, it still avoids the run-time look-ups to find the particular nodes, etc. In addition, other optimizations are possible.)

The approach presented in Delcher *et al.* [Delcher et al. 1996], like the Treitel/Genesereth [Treitel and Genesereth 1987] idea, does some work at data-input time (here "absorbing evidence") as a way to reduce the time required to answer the query (*i.e.*, compute the posterior probability), in the context of polytrees: rather than spending $O(1)$ time to absorb evidence then $O(N)$ time to compute the answer to a query (from an N-node polytree), their algorithm takes $O(\ln N)$ time to absorb each bit of evidence, then $O(\ln N)$ time to compute the probabilities.

As another tact: Appendix A.2 shows there are several known algorithms $\{QA_i\}$ for answering queries from a BN. While one such algorithm QA_A may be slow for a particular query, it is possible that another algorithm QA_B may be quite efficient for the same query. Moreover, different BNs can express the same distribution, and hence provide the same answers to all queries. Even if a specific QA_i algorithm is slow for a given query when using one net BN_α, that same algorithm may be efficient for this query, if it uses a different, but equivalent net BN_β.

We might therefore seek the "most efficient" $\langle \mathrm{B}, QA \rangle$ combination — *i.e.*, determine

- which algorithm (with which parameters) we should run,
- on which belief net (from the set of equivalent BNs)

to minimize the expected time to answer queries, where this "expected time" is averaged over the distribution of queries that will be encountered. (Hence, this task corresponds to the query-based model presented in [Greiner et al. 1997], but deals with efficiency, rather than acuracy.)

The search for the best algorithm/encoding does not need to be "blind", but can incorporate known results — *e.g.*, we could avoid "CutSet conditioning" if we show that it will always require more computations than any of the junction tree algorithms [Dechter 1996]; we can also use the empirical evaluations by Lepar and Shenoy [Lepar and Shenoy 1998], which suggests that the Shafer-Shenoy algorithm is more efficient than the Hugin algorithm, and that both improve on the original Lauritzen-Speigelhalter.

6.3 Improving Average *Performance*

The bulk of this paper has implicitly assumed we want our (expressive) reasoner to be as precise and correct as possible, while minimizing computational time. As noted above, an agent may need far less than this: *e.g.*, it may be able to act effectively given only a correct but vague answer (*e.g.*, whether some probability is $> 1/2$ or not), and it may survive if the answers it receives are correct at least (say) 90% of the time. Similarly, it may only need answers within 1 minute; getting answers ealier may not be advantageous, but getting then any, *later*, may be disasterous.

This motivates us to seek a reasoner whose "performance" is optimal, where "performance" is perhaps some combination of these { correctness, precision, expressiveness, efficiency } criteria. There are two obvious challenges here: The first is *determining this appropriate performance measure* — *i.e.*, defining a function that assigns a score to each reasoner, which can be used to compare a set of different reasoners (to decide which is best), and also to evaluate a single reasoner (to determine whether it is acceptable – *i.e.*, above a threshold). Of course, this criteria is extremely task-dependent.

As discussed in [Greiner and Elkan 1991], one approach is to first define a general "utility function" for a reasoner and query, that may combine the various criteria into a single score. One space of scores correspond to linear combinations of precision, correctness, expressiveness and time; *e.g.*, for a reasoner R and query q,

$$u(R,q) \quad = \quad \nu_{Prec} \times m_{Prec}(R,\, q) \;+\; \nu_{Time} \times m_{Time}(R,\, q) \;+\; \ldots$$

where each $m_\chi(R,\, q)$ measures the reasoner R's χ feature when dealing with the query q, and each $\nu_\chi \in \Re$ is a real value. The quality measure for a reasoner would then be its average utility score, over the distribution D_q of queries,

$$U(R) \quad = \quad E_{q \in D_q}[\, u(R,\, q)\,]$$

We would then prefer the reasoner that has the largest score. (We could, alternatively, let this individual query score be a combination of *thresholded* values. First associate with each query a set of "tolerances" required for time, precision, etc. — *e.g.*, perhaps the "tolerance query" $q = \langle\, P(\,\texttt{fire} \,|\, \texttt{smoke}, \neg\texttt{alarm}\,),\ \pm 0.25,\ \leq 2, \ldots \rangle$ means we need to know the probability of fire, given the evidence specified, to within ± 0.25, and get the answer within 2 seconds, etc. The utility score a reasoner R receives for this query q could then be defined as a weighted sum of precision and time: *e.g.*,

$$u(R,q) \quad = \quad \eta_{Prec} \times \delta[m_{Prec}(R,q) \leq 0.25] \;+\; \eta_{Time} \times \delta[m_{Time}(R,q) \leq\ 2] \;+\; \ldots$$

where $\delta[\cdots]$ is 1 if the condition is met, and 0 otherwise. Or we could use a more complicated function, that inter-relates these quantities perhaps using a more complicated combination function, etc. We assume the system designer has produced such a function, probably based on decision-theoretic considerations.

The second challenge is to identify the reasoner that is best, given this quality measure. As noted above, this will depend on the distribution of queries posed. While this may be unknown initially, we can use the standard techniques of sampling queries from the underlying distribution of queries to estimate the relevant information. Unfortunately, even if we knew the distribution precisely, we are left with the task of finding the reasoner that is optimal, with respect to this scoring function. In many situations, this is NP-hard or worse; see [Greiner 1991; Greiner and Schuurmans 1992]. The PALO learning system [Greiner 1996] was designed for this situation: This algorithm hill-climbs in the space of these reasoners, while collecting the samples it needs to estimate the quality of these reasoners. [Greiner 1996] proves, both theoretically and empirically, that this algorithm works efficiently, in part because it only seeks a local optimal, exploiting the local structure.

7. CONCLUSION

Many applications require a reasoner, often embedded as a component of some larger system. To be effective, this reasoner must be both reliable and efficient. This survey provides a collection of techniques that have been (or at least, should be) used to produce such a reasoner.

Of course, this survey provides only a sampling of the techniques. To keep the paper relatively short, we have had to skip many other large bodies of ideas. For example, we have not considered techniques that radically change the representation, perhaps by re-expressing the information as a neural net [Towell and Shavlik 1993], or by reasoning using the characteristic models of the theory [Kautz et al. 1993; Khardon and Roth 1994], rather than the theory itself.

We also chose to focus on techniques specific to reasoning, and so by-pass the huge inventory of techniques associated with improving the efficiency of computations, in general — including clever compilation techniques, and ways to exploit parallel algorithms. Needless to say, these ideas are also essential to producing reasoning algorithms that are as efficient as possible.

To summarize: we have overviewed a large variety of techniques for improving the effectiveness of a reasoning system, considering both sound logical reasoners (focussing on Horn clauses), and probabilistic reasoners (focussing on belief nets). In general, these techniques embody some (perhaps implicit) tradeoff, where the system designer is willing to sacrifice some desirable property (such as expressiveness, precision or correctness) to increase the system's efficiency.

We also discussed the idea of combining all of these measures into a single quality measure, and pointed to an algorithm that can "learn" the reasoning system that is optimal with respect to this measure.

Acknowledgement

RG gratefully acknowledges financial support from Siemens Corporate Research. We also thank Gary Kuhn for his many helpful comments.

APPENDIX

A. REASONING USING BELIEF NETS

A.1 Why is BN Inference hard?

In general, probailistic inference involve computing the posterior distribution of some query variable Q conditoned on some evidence $\vec{E} = \vec{e}$ — *i.e.*, computing $P(\,Q\,|\,\vec{E} = \vec{e}\,)$. Pearl [Pearl 1988] provided a linear-time algorithm for Belief Net inference for networks that have a poly-tree structure — where there is at most one (undirected) path connecting any pair of nodes. This algorithm simply propagates a few numbers from each node X to its immediate neighbors (children and parents), and then recurs in the simpler structure obtained by deleting this X.[21] Unfortunately, this idea does not work in general networks, which can contain (undirected) cycles. The basic problem, here, is that these (to be deleted) variables may induce dependencies among the other variables, which can make a difference in places far away in the network. To see this, consider asking for the (unconditional) probability that C is true $P(\,C = t\,)$ from the network shown in Figure 11. Here, a naïve (but incorrect) algorithm would first see that Z is true half of the time, which means that A and B are each true half of the time (as A is the same as Z and B is the same as $\neg Z$), which suggests that C, which is true only if both A and B are *t*rue, is *t*rue with probability $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$.

This is clearly wrong: A is true only when B is false, and vice versa. Hence, there is 0 chance that both A and B are true, and hence C is never *t*rue — *i.e.*, $P(\,C = t\,) = 0$. The issue, of course, is that we cannot simply propagate information from Z, and then forget about this source; there is instead a dependency between A and B, induced by their common parent Z. A related, but more complex, argument shows that such reasoning is, in fact NP-hard: here we can encode arbitrary 3SAT problem by including a node that represents the boolean formula, connected to nodes that represent the clauses (with a CPtable that insures that the formula is true iff all of the clauses is true), and the clause-nodes are each connected to nodes that represent the boolean variables (with CPtable that insure that the clause-node is true iff the associated boolean variables have the correct setting). Then the formula has a satisfying assignment iff the associated formula-variable has an unconditional probability strictly greater than 0; see [Cooper 1990].

A.2 Probabilistic Inference, using Belief Networks

There are two major categories of algorithms for computing posterior probabilities from general belief nets. Both rely on the observation that there is an efficient algorithm for computing arbitrary posterior probabilities for a poly-tree structure, and (perhaps implicitly) use this algorithm as a subroutine.

A.2.1 *Clustering.* The "clustering" trick involves converting the given belief network into an equivalent network that is a poly-tree, by merging various sets of nodes into associated "mega-nodes" — *e.g.*, consider transforming the network on the left side of Figure 12 to the network on the right side. Note that this new structure may be exponentially bigger than the the original belief net, as the CPtables for the

[21] The actual algorithm runs in two phases: first going from roots down to the leaves, and then going from the leaves up to the roots.

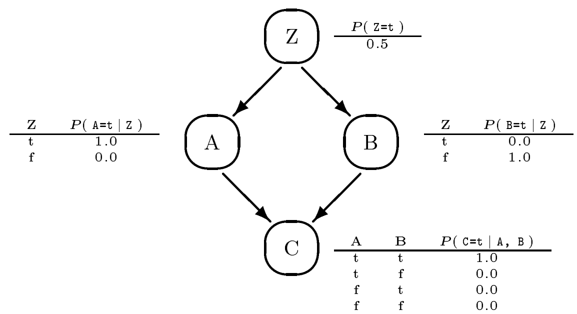


Fig. 11. Simple non-poly-tree Belief Network

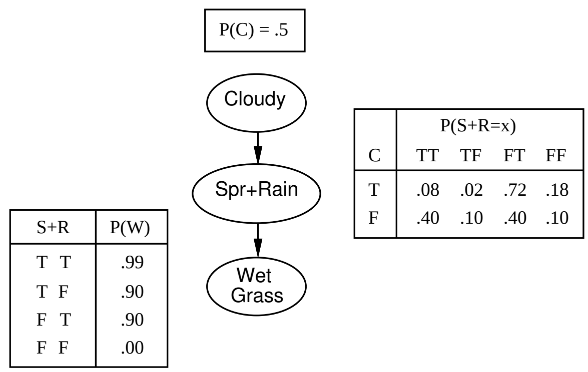


Fig. 12. Poly-Tree BN equivalent Figure 9 (taken from Figure 15.10 of [Russell and Norvig, 1995])

mega-node can be huge: if formed from the nodes $\mathcal{N} = \{N_1, \ldots N_k\}$, it will include a row for each combination of values from $\mathcal{N}$, and so (if each N_i is binary), it will have 2^k rows. (So while BN-inference will be linear in the size of the new poly-tree network, that size will be exponential in the size of the initial belief network.)

There can be many ways to form such clusters; see [Lauritzen and Spiegelhalter 1988] as well as implentation details for the Hugin system [Andersen et al. 1989; Jensen et al. 1990]. The general algorithm first "moralizes" the network (by connecting — aka "marrying" — the co-parents of each node) then triangulates the resulting graph, to form G'. It then forms a "junction tree" $T = \langle \mathcal{N}_T, \mathcal{A}_T \rangle$ — a tree structure whose nodes each correspond to the maximal cliques in G', and whose arcs $a = \langle n_1, n_2 \rangle \subset \mathcal{A}_T$ are each labeled with the nodes in the intersection between the G'-nodes that label n_1 and n_2. The algorithm then uses, in essense, the poly-tree algorithm to produce an answer to the original query.

This approach is also called the "junction tree" algorithm, or "clique tree" algorithm. See [Lepar and Shenoy 1998] for a specification of three of these algorithms, Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer.

Bucket Elimination: There are several simpler ways to understand this basic computation, including Li/B'Ambrosio's SPI [Li and D'Ambrosio 1994], Zhang/Poole's algorithm [Zhang and Poole 1994] and Dechter's bucket elimination [Dechter 1998].

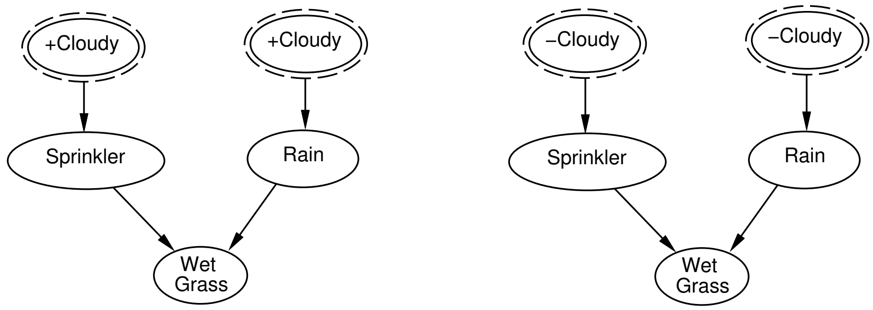


Fig. 13. Example of CutSet Conditioning (taken from Figure 15.11 of [Russell and Norvig, 1995])

We will focus on the third, which performs belief updating by storing the CPtables in a set of buckets — where the i^{th} bucket contains just those CPtables that involve only variables whose largest index is i. (Note that this algorithm requires an ordering of the nodes.) It then sequentially eliminates (read "marginalizes away") each variable X_i, updating the remaining buckets appropriately, by including in the appropriate (lower) bucket the marginals left after removing the dependencies on X_i.

Dechter shows that this relatively-straightforward algorithm is in fact doing the same computation as the general clustering algorithm, and has the same worst-case complexity. She also proves that (a small variant) of this algorithm corresponds to the poly-tree algorithm.

A.2.2 *Cut-set conditioning.* A "cut-set conditioning" algorithm also uses the poly-tree algorithm, but in a different manner. Given a belief net $\mathcal{B} = \langle \mathcal{N}, \mathcal{A}, \mathcal{CP} \rangle$ with nodes $\mathcal{N}$, and query $P(\,H \,|\, E\,)$, this algorithm first finds a subset $\mathcal{X} = \{X_1, \ldots, X_k\} \subset \mathcal{N}$, of nodes, such that

—$\mathcal{B}_{\mathcal{X}} \;=\; \mathcal{B} - \mathcal{X}$, the BN without $\mathcal{X}$, is a poly-tree, and

—$P(\,\mathcal{X} \,|\, E\,)$ is easy to compute

See [Suermondt and Cooper 1991]. It then exploits the equality

$$P(\,H \,|\, E\,) \quad = \quad \sum_{\vec{x}} P(\,H \,|\, E,\; \mathcal{X} = \vec{x}\,) \;\times\; P(\,\mathcal{X} = \vec{x} \,|\, E\,)$$

to answer the query $P(\,H \,|\, E\,)$. Note that each summard is each to compute, as $P(\,\mathcal{X} \,|\, E\,)$ is easy by construction, and $P(\,H \,|\, E,\; \mathcal{X} = \vec{x}\,)$ is a poly-tree computation. Figure 13 illustrates this construction.

The run-time for this algorithm is exponential in $|\mathcal{X}|$ (as it must sum over $\prod_{X \in \mathcal{X}} |\text{Domain}(X)|$ terms). However, its space requirement is linear in $|\mathcal{X}|$, as it need only maintain the running tally.

REFERENCES

AAAI. 1992. Workshop on approximation and abstraction of computational theories. Technical report, AAAI.

Abdelbar and Hedetniemi. 1998. Approximating MAPs for belief networks is NP-hard and other theorems. *Artificial Intelligence 102*, 21–38.

Alchourrón, C. E., Gärdenfors, P., and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic 50*, 510–30.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. 1989. HUGIN—a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Volume 2 (Detroit, Michigan, Aug. 1989), pp. 1080–1085. Morgan Kaufmann.

Arnborg, S. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — A survey. *BIT 25*, 2–33.

Azevedo, P. J. 1997. Magic sets with full sharing. *Journal of Logic Programming 30*, 3 (March), 223–237.

Bacchus, F., Grove, A., Halpern, J., and Koller, D. 1996. From statistical knowledge bases to degrees of belief. *Artificial Intelligence 87*, 75–143.

Barker, V. E., O'Connor, D. E., Bachant, J., and Soloway, E. 1989. Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM 32*, 3 (March), 298–318.

Beinlich, I. A., Suermondt, H. J., Chavez, R. M., and Cooper, G. F. 1989. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine,* London (Aug. 1989). Springer Verlag, Berlin.

Bobrow. 1980. Artificial intelligence. Special Issue on Non-Monotonic Logic.

Boros, E., Crama, Y., and Hammer, P. 1990. Polynomial-time inference of all valid implications for horn and related formulae. *Annals of Mathematics and Artificial Intelligence 1*, 21–32.

Bracewell, R. N. 1978. *The Fourier Transform and Its Applications.* McGraw Hill.

Brownston, L., Farrell, R., Kant, E., and Martin, N. 1986. *Programming Expert Systems in OPS-5.* Addison-Wesley, Reading, MA.

Chang, C.-L. and Lee, R. C.-T. 1973. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Inc., New York.

Charniak, E. 1991. Bayesian networks without tears. *AI Magazine 12*, 50–63.

Chaudhri, V. and Greiner, R. 1992. A formal analysis of solution caching. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence* (Vancouver, 1992).

Chow, C. K. and Lui, C. N. 1968. Aproximating discrete probability distributions with dependence trees. *IEEE Transactions on Informatoin Theory 14*, 3, 462–467.

Clark, K. 1978. Negation as failure. In H. Gallaire and J. Minker Eds., *Logic and Data Bases*, pp. 293–322. New York: Plenum Press.

Clocksin, W. F. and Mellish, C. S. 1981. *Programming in Prolog.* Springer-Verlag, New York.

Cooper, G. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence 42*, 2–3, 393–405.

Cousins, S., Chen, W., and Frisse, M. 1993. A tutorial introduction to stochastic simulation algorithms for belief networks. *Artificial Intelligence in Medicine*, 315–340.

Dagum and Luby. 1997. An optimal approximation algorithm for bayesian inference. *Artificial Intelligence 93*, 1–27.

Dagum, P. and Luby, M. 1993. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence 60*, 1 (March), 141–153.

Dalal, M. and Etherington, D. 1992a. Tractable approximate deduction using limited vocabulary. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence* (Vancouver, May 1992).

Dalal, M. and Etherington, D. W. 1992b. A hierarchy of tractable satisfiability problems. *Information Processing Letters 44*, 4 (Dec.), 173–180.

Darwiche, A. and Provan, G. M. 1996. Query dags: A practical paradigm for implementing belief network inference. In *Proc. Uncertainty in AI* (1996).

DARWICHE, A. AND PROVAN, G. M. 1997. A standard approach for optimizing belief network inference using query dags. In *Proc. Uncertainty in AI* (1997).

DECHTER, R. 1996. Topological parameters for time-space tradeoff. In E. HORVITZ AND F. JENSEN Eds., *Proc. Uncertainty in AI* (San Francisco, 1996), pp. 220–227. Morgan Kaufmann Publishers.

DECHTER, R. 1998. Bucket elimination: A unifying framework for probabilistic inference. In *Learning and Inference in Graphical Models* (1998).

DECHTER, R. AND RISH, I. 1997. A scheme for approximating probabilistic inference. In *Proc. Uncertainty in AI* (1997).

DELCHER, A. L., GROVE, A., KASIF, S., AND PEARL, J. 1996. Logarithmic-time updates and queries in probabilistic networks. *J. Artificial Intelligence Research 4*, 37–59.

DONINI, F. M., LENZERINI, M., NARDI, D., AND NUTT, W. 1997. The complexity of concept languages. *Information and Computation 134*, 1 (10 April), 1–58.

DOWLING, W. F. AND GALLIER, J. H. 1984. Linear time algorithms for testing the satisfiability of propositional horn formula. *Journal of Logic Programming 3*, 267–84.

DOYLE, J. AND PATIL, R. 1991. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence 48*, 3, 261–297.

DRAPER, D. L. AND HANKS, S. 1994. Localized partial evaluation of belief networks. In R. L. DE MANTARAS AND D. POOLE Eds., *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, July 1994), pp. 170–177. Morgan Kaufmann Publishers.

ELLMAN, T. 1993. Abstraction via approximate symmetry. In *IJCAI-93* (1993).

ENDERTON, H. B. 1972. *A Mathematical Introduction to Logic*. Academic Press, Inc., New York.

FELLER, W. 1966. *An Introduction to Probability Theory and Its Applications II*. John Wiley, New York.

FINDLER, N. V. Ed. 1979. *Associative Networks: Representation and Use of Knowledge by Computers* (New York, 1979). Academic Press.

FREUDER, E. C. 1996. *Principles and Practice of Constraint Programming*. Springer.

FRIEDMAN, N., GEIGER, D., AND GOLDSZMIDT, M. 1997. Bayesian network classifiers. *Machine Learning 29*, 131–?

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

GENESERETH, M. R. AND FIKES, R. E. 1992. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1 (June), Computer Science Department, Stanford University.

GENESERETH, M. R. AND NILSSON, N. J. 1987. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

GHAHRAMANI, Z. AND JORDAN, M. I. 1997. Factorial hidden markov models. *Machine Learning 29*, 245.

GINSBERG, M. 1987. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

GINSBERG, M. L. 1991. Knowledge interchange format: The KIF of death. Technical report, Stanford University.

GRATCH, J. AND DEJONG, G. 1996. A decision-theoretic approach to adaptive problem solving. *Artificial Intelligence 88*, 1–2, 365–396.

GREINER, R. 1991. Finding the optimal derivation strategy in a redundant knowledge base. *Artificial Intelligence 50*, 1, 95–116.

GREINER, R. 1996. PALO: A probabilistic hill-climbing algorithm. *Artificial Intelligence 83*, 1–2 (July), 177–204.

GREINER, R. 1999. Explanation-based learning. In R. WILSON AND F. KEIL Eds., *The Encyclopedia of Cognitive Science* (1999), pp. 301–303. MIT Press: A Bradford Book.

GREINER, R. AND ELKAN, C. 1991. Measuring and improving the effectiveness of representations. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (Sydney, Australia, Aug. 1991), pp. 518–524.

GREINER, R., GROVE, A., AND SCHUURMANS, D. 1997. Learning Bayesian nets that perform well. In *Uncertainty in Artificial Intelligence* (1997).

GREINER, R. AND SCHUURMANS, D. 1992. Learning useful horn approximations. In B. NEBEL, C. RICH, AND W. SWARTOUT Eds., *Proceedings of KR-92* (San Mateo, CA, Oct. 1992). Morgan Kaufmann.

HALPERN, J. Y. 1990. An analysis of first-order logics of probability. *Artificial Intelligence 46*, 3 (Dec.), 311–350.

HECKERMAN, D. 1991. *Probabilistic Similarity Networks*. MIT Press, Cambridge.

HECKERMAN, D. E. 1995. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research.

HERSKOVITS, E. H. AND COOPER, C. 1991. Algorithms for bayesian belief-network precomputation. In *Methods of Information in Medicine* (1991).

HOOS, H. H. AND STUTZLE, T. 1998. Evaluating las vegas algorithms – pitfalls and remedies. In *Proc. Uncertainty in AI* (San Francisco, 1998). Morgan Kaufmann Publishers.

HORVITZ, E. J., SUERMONDT, H. J., AND COOPER, G. F. 1989. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)* (Windsor, Ontario, 1989), pp. 182–193. Morgan Kaufmann.

JAAKKOLA, T. S. AND JORDAN, M. I. 1996a. Computing upper and lower bounds on likelihoods in intractable networks. In *Proc. Uncertainty in AI* (1996).

JAAKKOLA, T. S. AND JORDAN, M. I. 1996b. Recursive algorithms for approximating probabilities in graphical models. Technical Report 9604 (June), MIT Computational Cognitive Science.

JENSEN, F. V., LAURITZEN, S. L., AND OLESEN, K. G. 1990. Bayesian updating in causal probabilistic networks by local computations. *SIAM Journal on Computing 4*, 269–282.

KAUTZ, H., KEARNS, M., AND SELMAN, B. 1993. Reasoning with characteristic models. In *AAAI-93* (1993), pp. 34–39.

KAUTZ, H. AND SELMAN, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings AAAI96* (Menlo Park, Aug. 1996), pp. 1194–1201. AAAI Press / MIT Press.

KHARDON, R. AND ROTH, D. 1994. Reasoning with models. In *AAAI-94* (1994), pp. 1148–1153.

KJÆRULFF, U. 1994. Reduction of computational complexity in Bayesian networks through removal of weak dependences. In R. L. DE MANTARAS AND D. POOLE Eds., *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference* (Seattle, WA, 1994).

KOLLER, D., LEVY, A., AND PFEFFER, A. 1997. P-classic: A tractable probabilistic description logic. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)* (Providence, Rhode Island, Aug. 1997).

KOLLER, D. AND PFEFFER, A. 1997. Learning probabilities for noisy first-order rules. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)* (Nagoya, Japan, Aug. 1997).

KONDRAK, G. AND VAN BEEK, P. 1997. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence 89*, 365–387.

LAURITZEN, S. AND SPIEGELHALTER, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B 50*, 157–224. [Reprinted in [Shafer and Pearl 1990]].

LEPAR, V. AND SHENOY, P. P. 1998. A comparison of lauritzen-spiegelhalter, hugin, and shenoy-shafer architectures for computing marginals of probability distributions. In *Proc. Uncertainty in AI* (San Francisco, 1998). Morgan Kaufmann Publishers.

LEVESQUE, H. AND BRACHMAN, R. 1985. A fundamental tradeoff in knowledge representation and reasoning. In R. BRACHMAN AND H. LEVESQUE Eds., *Readings in Knowledge*

Representation (Los Altos, CA, 1985), pp. 41–70. Morgan Kaufmann Publishers, Inc.

Levesque, H. J. 1984. Foundations of a functional approach to knowledge representation. *Artificial Intelligence 23*, 155–212.

Li, Z. and D'Ambrosio, B. 1994. Efficient inference in bayes nets as a combinatorial optimization problem. *International Journal of Approximate Reasoning 11*, 1, 55–81.

Liu, C.-L. and Wellman, M. P. 1997. On state-space abstraction for anytime evaluation of bayesian network. *Sigart Bulletin 7*, 2.

Marek, W. and Truszczyński, M. 1989. Relating autoepistemic and default logics. In H. J. L. R. J. Brachman and R. Reiter Eds., *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning* (Toronto, Canada, May 1989), pp. 276–288. Morgan Kaufmann.

McAllester, D. A. 1989. *Ontic: A Knowledge Representation System for Mathematics.* MIT Press, Cambridge, Massachusetts.

McCarthy, J. 1977. Epistemological problems in artificial intelligence. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)* (Cambridge, Massachusetts, Aug. 1977). IJCAII.

McCarthy, J. 1980. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence 13*, 1,2 (April), 27–39.

McCune, W. and Wos, L. 1997. Otter—the CADE-13 competition incarnations. *Journal of Automated Reasoning 18*, 2 (April), 211–220.

Mengshoel, O. J. and Wilkins, D. 1997. Abstraction and aggregation in belief networks. In *AAAI97 Workshop on Abstractions, Decisions and Uncertainty* (1997).

Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. 1986. Example-based generalization: A unifying view. *Machine Learning 1*, 1, 47–80.

Moore, R. C. 1982. The role of logic in knowledge representation and commonsense reasoning. In D. Waltz Ed., *Proceedings of the National Conference on Artificial Intelligence* (Pittsburgh, PA, Aug. 1982), pp. 428–433. AAAI Press.

Nagel, E. and Newman, J. 1958. *Gödel's Proof*. New York University Press, New York.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swarton, W. R. 1991. Enabling technology for knowledge sharing. *AI Magazine*, 37–51.

Ngo, L. and Haddawy, P. 1995. Probabilistic logic programming and Bayesian networks. *Lecture Notes in Computer Science 1023*, 286–??

Nilsson, N. J. 1986. Probabilistic logic. *Artificial Intelligence 28*, 1 (Feb.), 71–88.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo.

Pelletier, F. J. 1986. THINKER. In J. H. Siekmann Ed., *Proceedings of the 8th International Conference on Automated Deduction*, Volume 230 of *LNCS* (Oxford, UK, July 1986), pp. 701–702. Springer.

Poole, D. 1993a. Average-case analysis of a search algorithm for estimating prior probabilities in bayesian networks with extreme probabilities. In *Proceedings IJCAI-93* (Chamberery, France, Aug. 1993), pp. 606–612.

Poole, D. 1993b. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence 64*, 81–129.

Poole, D., Goebel, R., and Aleliunas, R. 1987. Theorist: A logical reasoning system for default and diagnosis. In N. Cercone and G. McCalla Eds., *The Knowledge Frontier: Essays in the Representation of Knowledge* (New York, 1987), pp. 331–52. Springer Verlag.

Poole, D., Mackworth, A., and Goebel, R. 1998. *Computational Intelligence: A Logical Approach.* Oxford.

Pradhan, M. and Dagum, P. 1996. Optimal Monte Carlo estimation of belief network inference. In E. Horvitz and F. Jensen Eds., *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)* (San Francisco, Aug. 1996), pp. 446–453. Morgan Kaufmann Publishers.

PRADHAN, M., PROVAN, G., MIDDLETON, B., AND HENRION, M. 1994. Knowledge engineering for large belief networks. In R. L. DE MANTARAS AND D. POOLE Eds., *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, July 1994), pp. 484–490. Morgan Kaufmann Publishers.

RABINER, L. R. AND JUANG, B. H. 1986. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 4ff.

REITER, R. 1978a. Deductive question-answering on relational data bases. In H. GALLAIRE AND J. MINKER Eds., *Logic and Data Bases*, pp. 149–177. New York: Plenum Press.

REITER, R. 1978b. On closed world data bases. In H. GALLAIRE AND J. MINKER Eds., *Logic and Data Bases*, pp. 55–76. New York: Plenum Press.

REITER, R. 1980. Equality and domain closure in first-order databases. *Journal of the Association for Computing Machinery 27*, 2 (April), 235–49.

REITER, R. 1987. Nonmonotonic reasoning. In *Annual Review of Computing Sciences*, Volume 2, pp. 147–87. Palo Alto: Annual Reviews Incorporated.

ROBINSON, J. A. 1965. A machine-oriented logic based on the resolution principles. *JACM 12*, 23–41.

ROTH, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence Journal 82*, 1–2 (April), 273–302.

RUDNICKI, P. 1992. An overview of the Mizar project. Notes to a talk at the workshop on *Types for Proofs and Programs*.

SACERDOTI, E. D. 1973. Planning in a hierarchy of abstraction spaces. In *IJCAI-73* (1973).

SARKAR, S. AND MURTHY, I. 1996. Constructing efficient belief network structures with expert provided information. *IEEE Transactions on Knowledge and Data Engineering 8*, 1 (Feb.), 134–143.

SCOTT, A. C., CLAYTON, J. E., AND GIBSON, E. L. 1991. *A Practical guide to knowledge acquisition*. Addison-Wesley Pub Co., Reading, MA.

SELMAN, B. AND KAUTZ, H. 1996. Knowledge compilation and theory approximation. *Journal of the ACM 43*, 193–224.

SELMAN, B., LEVESQUE, H., AND MITCHELL, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (San Jose, July 1992), pp. 440–46.

SHAFER, G. AND PEARL, J. 1990. *Readings in Uncertain Reasoning*. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

SINGH, M. 1998. *Learning Belief Networks*. Ph. D. thesis, Department of Computer Science, University of Pennsylvania.

SMITH, D. E. AND GENESERETH, M. R. 1985. Ordering conjunctive queries. *Artificial Intelligence 26*, 2 (May), 171–215.

SMITH, D. E., GENESERETH, M. R., AND GINSBERG, M. L. 1986. Controlling recursive inference. *Artificial Intelligence 30*, 3, 343–89.

SMYTH, P., HECKERMAN, D., AND JORDAN, M. I. 1997. Probabilistic independence networks for hidden markov probability models. *Neural Computation 9*, 2, 227–269.

SRINIVAS, S. 1994. A probabilistic apprach to hierarchical model-based diagnosis. Technical Report Research Report No. KSL-94-14 (Feb.), Knowledge Systems Laboratory, Stanford University.

SUERMONDT, H. J. AND COOPER, G. F. 1991. Initialization for the method of conditioning in bayesian belief networks (research note). *Artificial Intelligence 50*, 1 (June), 83–94.

SWAMI, A. AND GUPTA, A. 1988. Optimization of large join queries. *SIGMOD Record (ACM Special Interest Group on Management of Data) 17*, 3 (Sept.), 8–17.

TOWELL, G. G. AND SHAVLIK, J. W. 1993. Extracting refined rules from knowledge-based neural networks. *Machine Learning 13*, 71–101.

TREITEL, R. J. AND GENESERETH, M. R. 1987. Choosing orders for rules. *Journal of Automated Reasoning 3*, 4 (Dec.), 395–432.

TURING, A. M. 1936. On computable numbers, with an application to the Entscheidungs problem. *Proc. London Math. Soc. 2*, 42, 230–265. (Corrections on volume 2(43):544–546).

Valiant, L. G. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing 8*, 3, 410–421.

van der Lans, R. F. 1989. *The SQL Standard: A complete reference*. Prentice Hall, Englewood Cliffs, NJ. Translated by Andrea Gray.

van Engelen, R. A. 1997. Approximating bayesian belief networks by arc removal. *IEEE PAMI 19*, 9 (Aug.).

Webb, G. I., Wells, J., and Zheng, Z. 1999. An experimental evaluation of integrating machine learning with knoweldge acquisition. *Machine Learning*, 5–21.

Weld, D. S. and de Kleer, J. 1990. *Readings in Qualitative Reasoning and Physical Systems*. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Wellman, M. 1994. Abstraction in probabilistic reasoning. Technical report, University of Michigan. Tutorial prepared for the *Summer Institute on Probability in AI*, Corvallis, OR, July 1994. See `http://ai.eecs.umich.edu/people/wellman/tut/Abstraction.html`.

Wellman, M. P. 1990. Fundamental concepts of qualitative probablistic networks. *Artificial Intelligence 44*, 3, 257–303.

Zhang, N. L. and Poole, D. 1994. A simple approach to bayesian network computations. In *Proc. of the 10th Canadian Conference on Artificial Intelligence* (Banff, Alberta, Canada, May 1994).

Zilberstein, S. 1993. Operational rationality through compilation of anytime algorithms. Technical Report CSD-93-743, University of California, Berkeley.

Zilberstein, S. 1996. Resource-bounded reasoning in intelligent systems. *ACM Computing Surveys 28*, 4es (Dec.), 15.