Published in final edited form as:
Ann Math Artif Intell. 2008 November 1; 54(1-3): 3–51.

Mixed deterministic and probabilistic networks

Robert Mateescu and
Electrical Engineering Department, California Institute of Technology, Pasadena, CA 91125, USA

Rina Dechter
Donald Bren School of Information and Computer Sciences, University of California Irvine, Irvine, CA 92697, USA

Robert Mateescu: mateescu@paradise.caltech.edu; Rina Dechter: dechter@ics.uci.edu

Abstract

The paper introduces *mixed networks*, a new graphical model framework for expressing and reasoning with probabilistic and deterministic information. The motivation to develop mixed networks stems from the desire to fully exploit the deterministic information (constraints) that is often present in graphical models. Several concepts and algorithms specific to belief networks and constraint networks are combined, achieving computational efficiency, semantic coherence and user-interface convenience. We define the semantics and graphical representation of mixed networks, and discuss the two main types of algorithms for processing them: inference-based and search-based. A preliminary experimental evaluation shows the benefits of the new model.

Keywords

Mixed network; Probabilistic information; Deterministic information; Graphical models; Automated reasoning; Inference; Search; AND/OR search

1 Introduction

Modeling real-life decision problems requires the specification of and reasoning with probabilistic and deterministic information. The primary approach developed in artificial intelligence for representing and reasoning with partial information under conditions of uncertainty is Bayesian networks. They allow expressing information such as “if a person has flu, he is likely to have fever.” Constraint networks and propositional theories are the most basic frameworks for representing and reasoning about deterministic information. Constraints often express resource conflicts frequently appearing in scheduling and planning applications, precedence relationships (e.g., “job 1 must follow job 2”) and definitional information (e.g., “a block is clear iff there is no other block on top of it”). Most often the feasibility of an action is expressed using a deterministic rule between the pre-conditions (constraints) and post-conditions that must hold before and after executing an action (e.g., STRIPS for classical planning).

The two communities of probabilistic networks and constraint networks matured in parallel with only minor interaction. Nevertheless some of the algorithms and reasoning principles that emerged within both frameworks, especially those that are graph-based, are quite related. Both

Correspondence to: Robert Mateescu, mateescu@paradise.caltech.edu.
Mathematics Subject Classifications (2000) 68T30 · 68T37 · 68T20 · 62F30 · 62F15

frameworks can be viewed as graphical models, a popular paradigm for knowledge representation in general.

Markov random fields (MRF) are another type of graphical model commonly used in statistical machine learning to describe joint probability distributions concisely. Their key property is that the graph is undirected, leading to isotropic or symmetric behavior. This is also the key difference compared to Bayesian networks, where a directed arc carries causal information. While the potential functions of an MRF are often assumed to be strictly positive, and are therefore not meant to handle deterministic relationships they can be easily extended to incorporate deterministic potentials with no need of any modification. Our choice however is the Bayesian network due to its appeal in semantic clarity and its representation of causal and directional information. In fact, our mixed networks can be viewed not only as a hybrid between probabilistic and deterministic information but also as a framework that permits causal information as well as symmetrical constraints.

Researchers within the logic-based and constraint communities have recognized for some time the need for augmenting deterministic languages with uncertainty information, leading to a variety of concepts and approaches such as non-monotonic reasoning, probabilistic constraint networks and fuzzy constraint networks. The belief networks community started more recently to look into mixed representation [15,24,32,38] perhaps because it is possible, in principle, to capture constraint information within belief networks [37].

In principle, constraints can be embedded within belief networks by modeling each constraint as a Conditional Probability Table (CPT). One approach is to add a new variable for each constraint that is perceived as its *effect* (child node) in the corresponding causal relationship and then to clamp its value to *true* [9,37]. While this approach is semantically coherent and complies with the acyclic graph restriction of belief networks, it adds a substantial number of new variables, thus cluttering the structure of the problem. An alternative approach is to designate one of the arguments of the constraint as a child node (namely, as its effect). This approach, although natural for functions (the arguments are the causes or parents and the function variable is the child node), is quite contrived for general relations (e.g., $x + 6 \neq y$). Such constraints may lead to cycles, which are disallowed in belief networks. Furthermore, if a variable is a child node of two different CPTs (one may be deterministic and one probabilistic) the belief network definition requires that they be combined into a single CPT.

The main shortcoming, however, of any of the above integrations is computational. Constraints have special properties that render them computationally attractive. When constraints are disguised as probabilistic relationships, their computational benefits may be hard to exploit. In particular, the power of constraint inference and constraint propagation may not be brought to bear.

Therefore, we propose a framework that combines deterministic and probabilistic networks, called *mixed network*. The identity of the respective relationships, as constraints or probabilities, will be maintained explicitly, so that their respective computational power and semantic differences can be vivid and easy to exploit. The mixed network approach allows two distinct representations: causal relationships that are directional and normally quantified by CPTs and symmetrical deterministic constraints. The proposed scheme's value is in providing: 1) semantic coherence; 2) user-interface convenience (the user can relate better to these two pieces of information if they are distinct); and most importantly, 3) computational efficiency. The results presented in this paper are based on the work in Dechter and Mateescu [16], Dechter and Larkin [15] and some part of Larkin and Dechter [26].

The paper is organized as follows: Section 2 provides background definitions and concepts for graphical models; Section 3 presents the framework of mixed networks, provides motivating

examples and extends the notions of conditional independence to the mixed graphs; Section 4 contains a review of inference and search algorithms for graphical models; Section 5 describes inference-based algorithms for mixed networks, based on Bucket Elimination; Section 6 describes search-based algorithms for mixed networks, based on AND/OR search spaces for graphical models; Section 7 contains the experimental evaluation of inference-based and AND/OR search-based algorithms; Section 8 describes related work and Section 9 concludes.

2 Preliminaries and background

Notations

A reasoning problem is defined in terms of a set of variables taking values on finite domains and a set of functions defined over these variables. We denote variables or subsets of variables by uppercase letters (e.g., X, Y, …) and values of variables by lower case letters (e.g., x, y, …). Sets are usually denoted by bold letters, for example $\mathbf{X} = \{X_1,\ldots, X_n\}$ is a set of variables. An assignment $(X_1 = x_1,\ldots, X_n = x_n)$ can be abbreviated as $\bar{x} = (\langle X_1, x_1\rangle, \ldots, \langle X_n, x_n\rangle)$ or $\bar{x} = (x_1,\ldots, x_n)$. For a subset of variables $\mathbf{Y}$, $D_{\mathbf{Y}}$ denotes the Cartesian product of the domains of variables in $\mathbf{Y}$. The projection of an assignment $\bar{x} = (x_1,\ldots, x_n)$ over a subset $\mathbf{Y}$ is denoted by $x_{\mathbf{Y}}$ or $x[\mathbf{Y}]$. We will also denote by $Y = y$ (or $\bar{y}$ for short) the assignment of values to variables in $\mathbf{Y}$ from their respective domains. We denote functions by letters f, g, h etc.

Graphical models

A graphical model $\mathcal{M}$ is a 3-tuple, $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}\rangle$, where: $\mathbf{X} = \{X_1,\ldots, X_n\}$ is a finite set of variables; $\mathbf{D} = \{D_1,\ldots, D_n\}$ is the set of their respective finite domains of values; $\mathbf{F} = \{f_1,\ldots, f_r\}$ is a set of non-negative real-valued discrete functions, each defined over a subset of variables $\mathbf{S}_i \subseteq \mathbf{X}$, called its scope, and denoted by $scope(f_i)$. A graphical model typically has an associated combination operator[1] $\otimes$, (e.g., $\otimes \in \{\prod, \Sigma, \bowtie\}$ (product, sum, join)). The graphical model represents the combination of all its functions: $\otimes_{i=1}^{r} f_i$. A graphical model has an associated primal graph that captures the structural information of the model:

Definition 1 (primal graph)—The *primal graph* of a graphical model is an undirected graph that has variables as its vertices and an edge connects any two variables that appear in the scope of the same function. We denote the primal graph by $G = (\mathbf{X}, E)$, where $\mathbf{X}$ is the set of variables and E is the set of edges.

Belief networks

A belief network is a graphical model $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P}\rangle$, where $\mathbf{G} = (\mathbf{X}, E)$ is a directed acyclic graph over the variables $\mathbf{X}$. The functions $\mathbf{P} = \{P_i\}$ are conditional probability tables $P_i = \{P(X_i \mid pa_i)\}$, where $pa_i = scope(P_i) \setminus \{X_i\}$ is the set of *parents* of X_i in $\mathbf{G}$. The primal graph of a belief network obeys the regular definition, and it can also be obtained as the *moral graph* of $\mathbf{G}$, by connecting all the nodes in every pa_i and then removing direction from all the edges. When the entries of the CPTs are "0" or "1" only, they are called *deterministic or functional CPTs.* The scope of P_i is also called the *family* of X_i (it includes X_i and its parents).

A belief network represents a probability distribution over $\mathbf{X}$ having the product form $P_{\mathcal{B}}(x) = P(x_1,\ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid x_{pa_i})$. An evidence set e is an instantiated subset of variables. The primary query over belief networks is to find the posterior probability of each single variable given some evidence e, namely to compute $P(X_i \mid e)$. Another important query is finding the *most probable explanation* (MPE), namely, finding a complete assignment to *all* the

[1]The combination operator can also be defined axiomatically [45].

variables having maximum probability given the evidence. A generalization of the MPE query is *maximum a posteriori hypothesis* (MAP), which requires finding the most likely assignment to a *subset* of hypothesis variables given the evidence.

Definition 2 (ancestral graph)—Given a directed graph G, *the ancestral graph relative to a subset of nodes* $\mathbf{Y}$ is the undirected graph obtained by taking the subgraph of G that contains $\mathbf{Y}$ and all their non-descendants, and then moralizing the graph.

Example 1

Figure 1a gives an example of a belief network over 6 variables, and Fig. 1b shows its moral graph. The example expresses the causal relationship between variables "Season" (A), "The configuration of an automatic sprinkler system" (B), "The amount of expected rain" (C), "The amount of necessary manual watering" (D), "How wet is the pavement" (F) and "Is the pavement slippery" (G). The belief network expresses the probability distribution $P(A, B, C, D, F, G) = P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(D|B, A) \cdot P(F|C, B) \cdot P(G|F)$.

Constraint networks

A constraint network is a graphical model $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$. The functions are constraints $\mathbf{C} = \{C_1, \ldots, C_t\}$. Each constraint is a pair $C_i = (\mathbf{S}_i, R_i)$, where $\mathbf{S}_i \subseteq \mathbf{X}$ is the scope of the relation R_i. The relation R_i denotes the allowed combination of values. The primary query over constraint networks is to determine if there exists a solution, namely, an assignment to all the variables that satisfies all the constraints, and if so, to find one. A constraint network represents the set of all its solutions. We sometimes denote the set of solutions of a constraint network $\mathcal{R}$ by $\varphi(\mathcal{R})$.

Example 2

Figure 2a shows a graph coloring problem that can be modeled as a constraint network. Given a map of regions, the problem is to color each region by one of the given colors {red, green, blue}, such that neighboring regions have different colors. The variables of the problem are the regions, and each one has the domain {red, green, blue}. The constraints are the relation "*different*" between neighboring regions. Figure 2b shows the constraint graph, and a solution (A=red, B=blue, C=green, D=green, E=blue, F=blue, G=red) is given in Fig. 2a.

Propositional theories

Propositional theories are special cases of constraint networks in which propositional variables which take only two values {*true*, *false*} or {1, 0}, are denoted by uppercase letters P, Q, R. Propositional *literals* (i.e., P, $\neg P$) stand for $P = true$ or $P = false$, and disjunctions of literals, or *clauses*, are denoted by α, β etc. For instance, $\alpha = (P \vee Q \vee R)$ is a clause. A *unit clause* is a clause that contains only one literal. The *resolution* operation over two clauses $(\alpha \vee Q)$ and $(\beta \vee \neg Q)$ results in a clause $(\alpha \vee \beta)$, thus eliminating Q. A formula φ in conjunctive normal form (CNF) is a set of clauses $\varphi = \{\alpha_1, \ldots, \alpha_t\}$ that denotes their conjunction. The set of *models* or *solutions* of a formula φ, denoted by $m(\varphi)$, is the set of all truth assignments to all its symbols that do not violate any clause.

3 Mixing probabilities with constraints

As shown in the previous section, graphical models can accommodate both probabilistic and deterministic information. Probabilistic information typically associates a strictly positive number with an assignment of variables, quantifying our expectation that the assignment may be realized. The deterministic information has a different semantics, annotating assignments with binary values, either *valid* or *invalid*. The mixed network allows probabilistic information

expressed as a belief network and a set of constraints to co-exist side by side and interact by giving them a coherent umbrella meaning.

3.1 Defining the mixed network

We give here the formal definition of the central concept of *mixed networks*, and discuss its relationship with the auxiliary network that hides the deterministic information through zero probability assignments.

Definition 3 (mixed networks)—Given a belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P} \rangle$ that expresses the joint probability $P_{\mathcal{B}}$ and given a constraint network $\mathcal{R} = \langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ that expresses a set of solutions $\rho(\mathcal{R})$ (or simply ρ), a mixed network based on $\mathcal{B}$ and $\mathcal{R}$ denoted $\mathcal{M}_{(\mathcal{B},\mathcal{R})} = \langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P}, \mathbf{C} \rangle$ is created from the respective components of the constraint network and the belief network as follows. The variables **X** and their domains are shared, (we could allow non-common variables and take the union), and the relationships include the CPTs in **P** and the constraints in **C**. The mixed network expresses the conditional probability $P_{\mathcal{M}}(\mathbf{X})$:

$$P_{\mathcal{M}}(\bar{x}) = \begin{cases} P_{\mathcal{B}}(\bar{x}|\bar{x} \in \rho), & if\ \bar{x} \in \rho \\ 0, & otherwise. \end{cases}$$

Clearly, $P_{\mathcal{B}}(\bar{x}|\bar{x} \in \rho) = \frac{P_{\mathcal{B}}(\bar{x})}{P_{\mathcal{B}}(\bar{x} \in \rho)}$. By definition, $P_{\mathcal{M}}(\bar{x}) = \prod_{i=1}^{n} P(x_i|\bar{x}_{pa_i})$ when $\bar{x} \in \rho$, and $P_{\mathcal{M}}(\bar{x}) = 0$ when $\bar{x} \notin \rho$. When clarity is not compromised, we will abbreviate $\langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P}, \mathbf{C} \rangle$ by $\langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{C} \rangle$ or $\langle \mathbf{X}, \mathbf{P}, \mathbf{C} \rangle$.

The auxiliary network: The deterministic information can be hidden through assignments having zero probability [37]. We now define the belief network that expresses constraints as pure CPTs.

Definition 4 (auxiliary network)—Given a mixed network $\mathcal{M}_{(\mathcal{B},\mathcal{R})}$, we define the auxiliary network $S_{(\mathcal{B},\mathcal{R})}$ to be a belief network constructed by augmenting $\mathcal{B}$ with a set of auxiliary variables defined as follows. For every constraint $C_i = (\mathbf{S}_i, R_i)$ in $\mathcal{R}$, we add the auxiliary variable A_i that has a domain of 2 values, "0" and "1". We also add a CPT over A_i whose parent variables are the set $\mathbf{S}_i$, defined by:

$$P(A_i=1|t) = \begin{cases} 1, & if\ t \in R_i \\ 0, & otherwise. \end{cases}$$

$S_{(\mathcal{B},\mathcal{R})}$ is a belief network that expresses a probability distribution P_S. It is easy to see that:

Proposition 1—Given a mixed network $\mathcal{M}_{(\mathcal{B},\mathcal{R})}$ and its associated auxiliary network $S = S_{(\mathcal{B},\mathcal{R})}$ then: $\forall \bar{x}\ P_{\mathcal{M}}(\bar{x}) = P_S(\bar{x} \mid A_1 = 1, \ldots, A_t = 1)$.

3.2 Queries over mixed networks

Belief updating, MPE and MAP queries can be extended to mixed networks straightforwardly. They are well defined relative to the mixed probability distribution $P_{\mathcal{M}}$. Since $P_{\mathcal{M}}$ is not well defined for inconsistent constraint networks, we always assume that the constraint network portion is consistent, namely it expresses a non-empty set of solutions. An additional relevant query over a mixed network is to find the probability of a consistent tuple relative to $\mathcal{B}$, namely

determining $P_{\mathcal{A}}(\bar{x} \in \rho(\mathcal{R}))$. It is called *CNF Probability Evaluation* or *Constraint Probability Evaluation (CPE)*. Note that the notion of evidence is a special type of constraint. We will elaborate on this next.

The problem of evaluating the probability of CNF queries over belief networks has various applications. One application is to network reliability described as follows. Given a communication graph with a source and a destination, one seeks to diagnose the failure of communication. Since several paths may be available, the reason for failure can be described by a CNF formula. Failure means that for all paths (conjunctions) there is a link on that path (disjunction) that fails. Given a probabilistic fault model of the network, the task is to assess the probability of a failure [40].

Definition 5 (CPE)—Given a mixed network $\mathcal{M}_{(\mathcal{B},\mathcal{R})}$, where the belief network is defined over variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and the constraint portion is a either a set of constraints $\mathcal{R}$ or a CNF formula ($\mathcal{R} = \varphi$) over a set of subsets $Q = \{Q_1, \ldots Q_r\}$, where $Q_i \subseteq \mathbf{X}$, the *constraint* (respectively *CNF) probability evaluation (CPE) task* is to find the probability $P_{\mathcal{A}}(\bar{x} \in \rho(\mathcal{R}))$, respectively $P_{\mathcal{A}}(\bar{x} \in m(\varphi))$, where $m(\varphi)$ are the models (solutions of φ).

Belief assessment conditioned on a constraint network or on a CNF expression is the task of assessing $P_{\mathcal{A}}(X|\varphi)$ for every variable X. Since $P(X|\varphi) = \alpha \cdot P(X \wedge \varphi)$ where α is a normalizing constant relative to X, computing $P_{\mathcal{A}}(X|\varphi)$ reduces to a CPE task over $\mathcal{A}$ for the query $((X = x) \wedge \varphi)$, for every x. More generally, $P(\varphi|\psi) = \alpha_{\varphi} \cdot P(\varphi \wedge \psi)$ where α_{φ} is a normalization constant relative to all the models of φ.

3.3 Examples of mixed networks

We describe now a few examples that can serve as motivation to combine probabilities with constraints in an efficient way. The first type of examples are real-life domains involving both type of information whereas some can conveniently be expressed using probabilistic functions and others as constraints. One such area emerged often in multi-agent environments. The second source comes from the need to process deterministic queries over a belief network, or accommodating disjunctive complex evidence which can be phrased as a propositional CNF sentence or as a constraint formula. As a third case, a pure belief network may involve deterministic functional CPTs. Those do not present semantical issues but can still be exploited computationally.

Java bugs: Consider the classical naive-Bayes model or, more generally, a two-layer network. Often the root nodes in the first layer are desired to be mutually exclusive, a property that can be enforced by *all-different* constraints. For example, consider a bug diagnostics system for a software application such as Java Virtual Machine that contains numerous bug descriptions. When the user performs a search for the relevant bug reports, the system outputs a list of bugs, in decreasing likelihood of it being the culprit of the problem. We can model the relationship between each bug identity and the key words that are likely to trigger this bug as a parent-child relationship of a two-layer belief network, where the bug identities are the root nodes and all the key words that may appear in each bug description are the child nodes. Each bug has a directed edge to each relevant keyword (See Fig. 3). In practice, it is common to assume that a problem is caused by only one bug and thus, the bugs on the list are mutually exclusive. We may want to express this fact using a not-equal relationship between all (or some of) the root nodes. We could have taken care of this by putting all the bugs in one node. However, this would cause a huge inconvenience, having to express the conditional probability of each key word given each bug, even when it is not relevant. Java bug database contains thousands of bugs. It is hardly sensible to define a conditional probability table of that size. Therefore, in

the mixed network framework we can simply add one not-equal constraint over all the root variables.

Class scheduling: Another source of examples is reasoning about the behavior of an agent. Consider the problem of scheduling classes for students. A relevant knowledge base can be built from the point of view of a student, of the administration or of the faculty. Perhaps, the same knowledge base can serve these multiple reasoning perspectives. The administration (e.g., the chair) tries to schedule the classes so as to meet the various requirements of the students (allow enough classes in each quarter for each concentration), while faculty may want to teach their classes in a particular quarter to maximize (or minimize) the attendance or to better allocate their research vs. teaching time throughout the academic year.

In Fig. 4 we demonstrate a scenario with 3 classes and 1 student. The variables corresponding to the student S_i can be repeated to model all the students, but we keep the figure simple. The dotted lines indicate deterministic relationships, and the solid arrows indicate probabilistic links. The variables are: *Enrolled*(S_i, C_j) meaning "student S_i takes course C_j"; *Grade*(S_i, C_j) denoting the grade (performance) of student S_i in course C_j; *Past-Grade*(S_i, C_j) is the past performance (grade) of student S_i in C_j (if the class was taken); the variable *Professor*(C_j) denotes the professor who teaches the class C_j in the current quarter, and *Type*(S_i) stands for a collection of variables denoting student S_i's characteristics (his strengths, goals and inclinations, time in the program etc.). If we have a restriction on the number of students that can take a class, we can impose a unary constraint (*Class-Size*(C_i) ≤ 10). For each student and for each class, we have a CPT for *Grade*(S_i, C_j) with the parent nodes *Enrolled*(S_i, C_j), *Professor* (C_j) and *Type*(S_i). We then have constraints between various classes such as *Enrolled*(S_i, C_1) and *Enrolled*(S_i, C_2) indicating that both cannot be taken together due to scheduling conflicts. We can also have all-different constraints between pairs of *Professor*(C_j) since the same professor may not teach two classes even if those classes are not conflicting (for clarity we do not express these constraints in Fig. 4). Finally, since a student may need to take at least 2 and at most 3 classes, we can have a variable *Number-of-Classes*(S_i) that is the number of classes taken by the student. If a class is a prerequisite to another we can have a constraint that limits the enrollment in the follow-up class. For example, in the figure C_5 is a prerequisite to both C_2 and C_3, and therefore *Enrolled*(S_1, C_2) and *Past-Grade*(S_1, C_5) are connected by a constraint. If the past grade is not satisfactory, or missing altogether (meaning the class was not taken), then the enrollment in C_2 and C_3 is forbidden. The primary task for this network is to find an assignment that satisfies all the preferences indicated by the professors and students, while obeying the constraints. If the scheduling is done once at the beginning of the year for all the three quarters, the probabilistic information related to *Grade*(S_i, C_i) can be used to predict the eligibility to enroll in follow-up classes during the same year.

Retail data analysis: A real life example is provided by the problem of analyzing large retail transaction data sets. Such data sets typically contain millions of transactions involving several hundred product categories. Each attribute indicates whether a customer purchased a particular product category or not. Examples of these product attributes are `sports-coat`, `rain-coat`, `dress-shirt`, `tie`, etc. Marketing analysts are interested in posing queries such as "how many customers purchased a coat and a shirt and a tie?" In Boolean terms this can be expressed (for example) as the CNF query `(sports-coat rain-coat) (dress-shirt casual-shirt) tie`. A query expressed as a conjunction of such clauses represents a particular type of prototypical transaction (particular combination of items) and the focus is on discovering more information about customers who had such a combination of transactions. We can also have ad probabilistic information providing prior probabilities for some categories, or probabilistic dependencies between them yielding a belief network. The queries can then become the CNF probability evaluation problem.

Genetic linkage analysis: Genetic linkage analysis is a statistical method for mapping genes onto a chromosome, and determining the distance between them [34]. This is very useful in practice for identifying disease genes. Without going into the biology details, we briefly describe how this problem can be modeled as a reasoning task in a mixed network.

Figure 5a shows the simplest pedigree, with two parents (denoted by 1 and 2) and an offspring (denoted by 3). Square nodes indicate males and circles indicate females. Figure 5c shows the usual belief network that models this small pedigree for two particular loci (locations on the chromosome). There are three types of variables, as follows. The *G* variables are the genotypes (the values are the specific alleles, namely the forms in which the gene may occur on the specific locus), the *P* variables are the phenotypes (the observable characteristics). Typically these are evidence variables, and for the purpose of the graphical model they take as value the specific unordered pair of alleles measured for the individual. The *S* variables are selectors (taking values 0 or 1). The upper script *p* stands for paternal, and the *m* for maternal. The first subscript number indicates the individual (the number from the pedigree in Fig. 5a), and the second subscript number indicates the locus. The interactions between all these variables are indicated by the arcs in Fig. 5c.

Due to the genetic inheritance laws, many of these relationships are actually deterministic. For example, the value of a selector variable determines the genotype variable. Formally, if *a* is the father and *b* is the mother of *x*, then:

$$G^p_{x,j}=\begin{cases} G^p_{a,j}, & \text{if } S^p_{x,j}=0 \\ G^m_{a,j}, & \text{if } S^p_{x,j}=1 \end{cases} \quad \text{and} \quad G^m_{x,j}=\begin{cases} G^p_{b,j}, & \text{if } S^m_{x,j}=0 \\ G^m_{b,j}, & \text{if } S^m_{x,j}=1 \end{cases}$$

The CPTs defined above are in fact deterministic, and can be captured by a constraint, depicted graphically in Fig. 5b. he only real probabilistic information appears in the CPTs of two types of variables. The first type are the selector variables $S^p_{i,j}$ and $S^m_{i,j}$. The second type are the founders, namely the individuals having no parents in the pedigree, for example $G^p_{1,2}$ and $G^m_{1,2}$ in our example.

Genetic linkage analysis is an example where we do not "need" the mixed network formulation, because the constraints are "causal" and can naturally be part of the directed model. However, it is an example of a belief network that contains many deterministic or functional relations that can be exploited as constraints. The typical reasoning task is equivalent to belief updating or computing the probability of the evidence, or to maximum probable explanation, which can be solved by inference-based or search-based approaches as we will discuss in the following sections.

3.4 Processing probabilistic networks with determinism by CPE queries

In addition to the need to express non-directional constraints, in practice pure belief networks often have hybrid probabilistic and deterministic CPTs as we have seen in the linkage example. Additional example networks appear in medical applications [36], in coding networks [29] and in networks having CPTs that are *causally independent* [21]. Using constraint processing methods can potentially yield a significant computational benefit and we can address it using CPE queries as explained next.

Belief assessment in belief networks having determinism can be translated to a CPE task over a mixed network. The idea is to collect together all the deterministic information appearing in the functions of **P**, namely to extract the deterministic information from the CPTs, and then

transform it all to one CNF or a constraint expression that will be treated as a constraint network part relative to the original belief network. Each entry in a mixed CPT $P(X_i|pa_i)$, having $P(x_i|x_{pa_i}) = 1$ (x is a tuple of variables in the family of X_i), can be translated to a constraint (not allowing tuples with zero probability) or to clauses $x_{pa_i} \rightarrow x_i$, and all such entries constitute a conjunction of clauses or constraints.

Let $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P} \rangle$ be a belief network having determinism. Given evidence e, assessing the posterior probability of a single variable X given evidence e requires computing $P(X|e) = \alpha \cdot P(X \wedge e)$. Let $cl(P)$ be the clauses extracted from the mixed CPTs. The deterministic portion of the network is now $cl(P)$. We can write: $P((X = x) \wedge e) = P((X = x) \wedge e \wedge cl(P))$. Therefore, to evaluate the belief of $X = x$ we can evaluate the probability of the CNF formula $\varphi = ((X = x) \wedge e \wedge cl(P))$ over the original belief network. In this case redundancy is allowed because expressing a deterministic relation both probabilistically and as a constraint is semantically valid.

3.5 Mixed graphs as I-maps

In this section we define the *mixed graph* of a mixed network and an accompanying separation criterion, extending d-separation [37]. We show that a mixed graph is a minimal I-map (independency map) of a mixed network relative to an extended notion of separation, called *dm-separation*.

Definition 6 (mixed graph)—Given a mixed network $\mathcal{M}_{(\mathcal{B},\mathcal{R})}$, the mixed graph $G_{\mathcal{M}} = (G, D)$ is defined as follows. Its nodes correspond to the variables appearing either in $\mathcal{B}$ or in $\mathcal{R}$, and the arcs are the union of the undirected arcs in the constraint graph D of $\mathcal{R}$, and the directed arcs in the directed acyclic graph G of the belief network $\mathcal{B}$. The moral mixed graph is the moral graph of the belief network union the constraint graph.

The notion of d-separation in belief networks is known to capture conditional independence [37]. Namely any d-separation in the directed graph corresponds to a conditional independence in the corresponding probability distribution defined over the directed graph. Likewise, an undirected graph representation of probabilistic networks (i.e., Markov random fields) allows reading valid conditional independence based on undirected graph separation.

In this section we define a *dm-separation* of mixed graphs and show that it provides a criterion for establishing minimal I-mapness for mixed networks.

Definition 7 (ancestral mixed graph)—Given a mixed graph $G_{\mathcal{M}} = (G, D)$ of a mixed network $\mathcal{M}_{(\mathcal{B},\mathcal{R})}$ where G is the directed acyclic graph of $\mathcal{B}$, and D is the undirected constraint graph of $\mathcal{R}$, the ancestral graph of X in $G_{\mathcal{M}}$ is the graph D union the ancestral graph of X in G.

Definition 8 (dm-separation)—Given a mixed graph, $G_{\mathcal{M}}$ and given three subsets of variables **X**, **Y** and **Z** which are disjoint, we say that **X** and **Y** are dm-separated given **Z** in the mixed graph $G_{\mathcal{M}}$, denoted $< \mathbf{X}, \mathbf{Z}, \mathbf{Y} >_{dm}$, iff in the ancestral mixed graph of $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$, all the paths between **X** and **Y** are intercepted by variables in **Z**.

The following theorem follows straightforwardly from the correspondence between mixed networks and auxiliary networks.

Theorem 1 (I-map)—Given a mixed network $\mathcal{M} = \mathcal{M}_{(\mathcal{B},\mathcal{R})}$ and its mixed graph $G_{\mathcal{M}}$ then $G_{\mathcal{M}}$ is a minimal I-map of $\mathcal{M}$ relative to dm-separation. Namely, if $<\mathbf{X}, \mathbf{Z}, \mathbf{Y}>_{\mathrm{dm}}$ then $P_{\mathcal{M}}(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) = P_{\mathcal{M}}(\mathbf{X}|\mathbf{Z})$ and no arc can be removed while maintaining this property.

Proof: Assuming $<\mathbf{X}, \mathbf{Z}, \mathbf{Y}>_{dm}$ we should prove $P_{\mathcal{M}}(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) = P_{\mathcal{M}}(\mathbf{X}|\mathbf{Z})$. Namely, we should prove that $P_S(\mathbf{X}|\mathbf{Y}, \mathbf{Z}, A = 1) = P_S(\mathbf{X}|\mathbf{Z}, A = 1)$, when $S = S_{(\mathcal{B},\mathcal{R})}$, and $A = 1$ is an abbreviation to assigning all auxiliary variables in S the value 1 (Proposition 1). Since $S = S_{(\mathcal{B},\mathcal{R})}$ is a regular belief network we can use the ancestral graph criterion to determine d-separation. It is easy to see that the ancestral graph of the directed graph of S given $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z} \cup A$ is identical to the corresponding ancestral mixed graph (if we ignore the edges going into the evidence variables A), and thus dm-separation translates to d-separation and provides a characterization of I-mapness of mixed networks. The minimality of mixed graphs as I-maps follows from the minimality of belief networks relative to d-separation applied to the auxiliary network.

Example 3: Figure 6a shows a regular belief network in which X and Y are d-separated given the empty set. If we add a constraint R_{PQ} between P and Q, we obtain the mixed network in Fig. 6b. According to dm-separation X is no longer independent of Y, because of the path $XPQY$ in the ancestral graph. Figure 6c shows the auxiliary network, with variable A assigned to 1 corresponding to the constraint between P and Q. D-separation also dictates a dependency between X and Y.

We will next see the first virtue of "mixed" network when compared with the "auxiliary" network. Namely, it will allow the constraint network to be processed by any constraint propagation algorithm to yield another, equivalent, well defined, mixed network.

Definition 9 (equivalent mixed networks)—Two mixed networks defined on the same set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and the same domains, $D_1, \ldots, D_n$, denoted by $\mathcal{M}_1 = \mathcal{M}_{(\mathcal{B}_1,\mathcal{R}_1)}$ and $\mathcal{M}_2 = \mathcal{M}_{(\mathcal{B}_2,\mathcal{R}_2)}$, are equivalent iff they are equivalent as probability distributions, namely iff $P_{\mathcal{M}_1} = P_{\mathcal{M}_2}$ (see Definition 3).

Proposition 2—If $\mathcal{R}_1$ and $\mathcal{R}_2$ are equivalent constraint networks (i.e., they have the same set of solutions), then for any belief network $\mathcal{B}$, $\mathcal{M}_{(\mathcal{B},\mathcal{R}_1)}$ is equivalent to $\mathcal{M}_{(\mathcal{B},\mathcal{R}_2)}$

Proof: The proof follows directly from Definition 3.

The following two propositions show that if we so desire, we can avoid redundancy or exploit redundancy by moving deterministic relations from $\mathcal{B}$ to $\mathcal{R}$ or vice versa.

Proposition 3—Let $\mathcal{B}$ be a belief network and $P(x|pa_x)$ be a deterministic CPT that can be expressed as a constraint $C(x, pa_x)$. Let $\mathcal{B}_1 = \mathcal{B} \setminus P(x|pa_x)$. Then $\mathcal{M}_{(\mathcal{B},\phi)} = \mathcal{M}_{(\mathcal{B}_1,C)} = \mathcal{M}_{(\mathcal{B},C)}$.

Proof: All three mixed networks $\mathcal{M}_{(\mathcal{B},\phi)}$, $\mathcal{M}_{(\mathcal{B}_1,C)}$ and $\mathcal{M}_{(\mathcal{B},C)}$ admit the same set of tuples of strictly positive probability. Furthermore, the probabilities of the solution tuples are defined by all the CPTs of $\mathcal{B}$ except $P(x|pa_x)$. Therefore, the three mixed networks are equivalent.

Corollary 1—Let $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P} \rangle$ be a belief network and $\mathbf{F}$ a set of constraints extracted from $\mathbf{P}$. Then $\mathcal{M}_{(\mathcal{B},\phi)} = \mathcal{M}_{(\mathcal{B},\mathbf{F})}$.

In conclusion, the above corollary shows one advantage of looking at mixed networks rather than at auxiliary networks. Due to the explicit representation of deterministic relationships, notions such as inference and constraint propagation are naturally defined and are exploitable in mixed networks.

4 Inference and search for graphical models

In this section we review the two main algorithmic approaches for graphical models: inference and search. Inference methods process the available information, derive and record new

information (typically involving one less variable), and proceed in a dynamic programming manner until the task is solved. Search methods perform reasoning by conditioning on variable values and enumerating the entire solution space. In Sections 5 and 6 we will show how these methods apply for mixed deterministic and probabilistic networks.

4.1 Inference methods

Most inference methods assume an ordering of the variables, that dictates the order in which the functions are processed. The notion of *induced width* or *treewidth* is central in characterizing the complexity of the algorithms.

Induced graphs and induced width: An *ordered graph* is a pair (G, d) where G is an undirected graph, and $d = X_1, \ldots, X_n$ is an ordering of the nodes. The *width of a node* in an ordered graph is the number of the node's neighbors that precede it in the ordering. The *width of an ordering d*, denoted $w(d)$, is the maximum width over all nodes. The *induced width of an ordered graph*, $w^*(d)$, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X is processed, all its preceding neighbors are connected. The *induced width of a graph*, w^*, is the minimal induced width over all its orderings. The *treewidth* of a graph is the minimal induced width over all orderings.

Bucket elimination: As an example of inference methods, we will give a short review of Bucket Elimination, which is a unifying framework for variable elimination algorithms applicable to probabilistic and deterministic reasoning [5,12,18,47]. The input to a bucket-elimination algorithm is a knowledge-base theory specified by a set of functions or relations (e.g., clauses for propositional satisfiability, constraints, or conditional probability tables for belief networks). Given a variable ordering, the algorithm partitions the functions (e.g., CPTs or constraints) into buckets, where a function is placed in the bucket of its latest argument in the ordering. The algorithm processes each bucket, from last to first, by a variable elimination procedure that computes a new function that is placed in an earlier (lower) bucket. For belief assessment, when the bucket does not have an observed variable, the bucket procedure computes the product of all the probability tables and sums over the values of the bucket's variable. Observed variables are independently assigned to each function and moved to the corresponding bucket, thus avoiding the creation of new dependencies. Algorithm 1 shows *Elim-Bel*, the bucket-elimination algorithm for belief assessment. The time and space complexity of such algorithms is exponential in the induced width w^*. For more information see Dechter [13].

Algorithm 1: ELIM-BEL

input : A belief network $\mathcal{B} = \{P_1,\ldots,P_n\}$; an ordering of the variables, d; observations e.

output : The updated belief $P(X_1|e)$, and $P(e)$.

1 Partition $\mathcal{B}$ into $bucket_1, \ldots, bucket_n$ // Initialize

2 **for** $p \leftarrow n$ **down to** 1 **do** // Backward

Let $\lambda_1, \lambda_2, \ldots, \lambda_j$ be the functions in $bucket_p$

if $bucket_p$ *contains evidence* $X_p = x_p$ **then**

for $i \leftarrow 1$ **to** j **do**

Assign $X_p \leftarrow x_p$ in λ_i

Move λ_i to the bucket of its latest variable

else

Generate $\lambda^p = \sum_{X_p} \prod_{i=1}^{j} \lambda_i$

Add λ^p to the bucket of its latest variable

3 **return** $P(X_1|e)$ by normalizing the product in $bucket_1$, and $P(e)$ as the normalizing factor.

4.2 AND/OR search methods

As a framework for search methods, we will use the recently proposed AND/OR search space framework for graphical models [17]. The usual way to do search (called here *OR search*) is to instantiate variables in a static or dynamic order. In the simplest case this defines a search tree, whose nodes represent states in the space of partial assignments, and the typical depth first (DFS) algorithm searching this space would require linear space. If more space is available, then some of the traversed nodes can be cached, and retrieved when encountered again, and the DFS algorithm would in this case traverse a graph rather than a tree.

The traditional OR search space however does not capture any of the structural properties of the underlying graphical model. Introducing *AND* nodes into the search space can capture the structure of the graphical model by decomposing the problem into independent subproblems. The *AND/OR search space* is a well known problem solving approach developed in the area of heuristic search, that exploits the problem structure to decompose the search space. The states of an AND/OR space are of two types: *OR* states which usually represent alternative ways of solving the problem (different variable values), and *AND* states which usually represent problem decomposition into subproblems, all of which need to be solved. We will next present the AND/OR search space for a general *graphical model* which in particular applies to mixed networks. The AND/OR search space is guided by a pseudo tree that spans the original graphical model.

Definition 10 (pseudo tree): *A pseudo tree* of a graph $G = (\mathbf{X}, E)$ is a rooted tree $\mathcal{T}$ having the same set of nodes $\mathbf{X}$, such that every edge in E is a backarc in $\mathcal{T}$ (i.e., it connects nodes on the same path from root).

Given a reasoning graphical model $\mathcal{M}$ (e.g., belief network, constraint network, influence diagram) its primal graph G and a pseudo tree $\mathcal{T}$ of G, the associated AND/OR tree is defined as follows [17].

Definition 11 (AND/OR search tree of a graphical model): Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, its primal graph G and a pseudo tree $\mathcal{T}$ of G, the associated AND/OR search tree has alternating levels of OR and AND nodes. The OR nodes are labeled X_i and correspond to variables. The AND nodes are labeled $\langle X_i, x_i \rangle$ (or simply x_i) and correspond to value assignments. The structure of the AND/OR search tree is based on $\mathcal{T}$. The root is an OR node labeled with the root of $\mathcal{T}$. The children of an OR node X_i are AND nodes labeled with assignments $\langle X_i, x_i \rangle$ (or x_i) that are consistent with the assignments along the path from the root. The children of an AND node $\langle X_i, x_i \rangle$ are OR nodes labeled with the children of X_i in $\mathcal{T}$. A *solution subtree* of an AND/OR search graph is a subtree that: (1) contains the root node of the AND/OR graph; (2) if an OR node is in the subtree, then one and only one of its children is in the subtree; (3) if an AND node is in the subtree, then all of its children are in the subtree; (4) the assignment corresponding to the solution subtree is consistent with respect to the graphical model (i.e., it has a non-zero value with respect to the functions of the model).

Example 4: Figure 7 shows an example of an AND/OR search tree. Figure 7a shows a graphical model defined by four functions, over binary variables, and assuming all tuples are consistent. When some tuples are inconsistent, some of the paths in the tree do not exists. Figure 7b gives the pseudo tree that guides the search, from top to bottom, as indicated by the arrows. The dotted arcs are backarcs from the primal graph. Figure 7c shows the AND/OR search tree, with

the alternating levels of OR (circle) and AND (square) nodes, and having the structure indicated by the pseudo tree. In this case we assume that all tuples are consistent.

The AND/OR search tree for a graphical model specializes the notion of AND/OR search spaces for state-space models as defined in Nilsson [33]. The AND/OR search tree can be traversed by a depth first search algorithm, thus using linear space. It was already shown [4, 6,10,16,17,19] that:

Theorem 2: Given a graphical model $\mathcal{M}$ and a pseudo tree $\mathcal{T}$ of depth m, the size of the AND/OR search tree based on $\mathcal{T}$ is $O(n\ k^m)$, where k bounds the domains of variables. A graphical model of treewidth w* has a pseudo tree of depth at most w* log n, therefore it has an AND/OR search tree of size $O(n\ k^{w^* \log n})$.

The AND/OR search tree expresses the set of all possible assignments to the problem variables (all solutions). The difference from the traditional OR search space is that a solution is no longer a path from root to a leaf, but rather a subtree. The AND/OR search tree may contain nodes that root identical subproblems. These nodes are said to be *unifiable*. When unifiable nodes are merged, the search space becomes a graph. Its size becomes smaller at the expense of using additional memory by the search algorithm. The depth first search algorithm can therefore be modified to cache previously computed results, and retrieve them when the same nodes are encountered again.

Some unifiable nodes can be identified based on their *contexts*. We can define graph based contexts for the variables by expressing the set of ancestor variables in the pseudo tree that completely determine a conditioned subproblem

Definition 12 (context): Given a pseudo tree $\mathcal{T}$ of an AND/OR search space, $context(X) = [X_1 \dots X_p]$ is the set of ancestors of X in $\mathcal{T}$, ordered descendingly, that are connected in the primal graph to X or to descendants of X.

Definition 13 (context minimal AND/OR graph): Given an AND/OR search graph, two OR nodes n_1 and n_2 are *context unifiable* if they have the same variable label X and the assignments of their contexts are identical. Namely, if π_1 is the partial assignment of variables along the path to n_1, and π_2 is the partial assignment of variables along the path to n_2, then their restriction to the context of X is the same: $\pi_1|_{context(X)} = \pi_2|_{context(X)}$. The *context minimal* AND/OR graph is obtained from the AND/OR search tree by merging all the context unifiable OR nodes.

It was already shown [4,10,17] that:

Theorem 3: Given a graphical model $\mathcal{M}$, its primal graph G and a pseudo tree $\mathcal{T}$, the size of the context minimal AND/OR search graph based on $\mathcal{T}$ is $O(n\ k^{w^*_{\mathcal{T}}(G)})$, where $w^*_{\mathcal{T}}(G)$ is the induced width of G over the depth first traversal of $\mathcal{T}$, and k bounds the domain size.

Example 5: For Fig. 8 we refer to the model in Fig. 7a, assuming that all assignments are valid and that variables take binary values. Figure 8a shows the pseudo tree derived from ordering $d = (A, B, E, C, D)$. The context of each node appears in square brackets, and the dotted arcs are backarcs. Figure 8b shows the context minimal AND/OR graph.

4.2.1 Weighted AND/OR graphs—In Dechter and Mateescu [17] it was shown how the probability distribution of a given belief network can be expressed using AND/OR graphs, and how queries of interest, such as computing the posterior probability of a variable or the probability of the evidence, can be computed by a depth-first search traversal. All we need is to annotate the OR-to-AND arcs with weights derived from the relevant CPTs, such that the

product of weights on the arc of any solution subtree is equal to the probability of that solution according to the belief network.

Formally, given a belief network $\mathcal{B} = \langle \mathbf{X}, \mathbf{D}, \mathbf{G}, \mathbf{P} \rangle$ and a pseudo tree $\mathcal{T}$, the *bucket* of X_i relative to $\mathcal{T}$, denoted $B_{\mathcal{T}}(X_i)$, is the set of functions whose scopes contain X_i and are included in $path_{\mathcal{T}}(X_i)$, which is the set of variables from the root to X_i in $\mathcal{T}$. Namely, $B_{\mathcal{T}}(X_i) = \{P_j \in \mathbf{P} | X_i \in scope(P_j), scope(P_j) \subseteq path_{\mathcal{T}}(X_i)\}$. A CPT belongs to the bucket of a variable X_i iff its scope has just been fully instantiated when X_i was assigned. Combining the values of all functions in the bucket, for the current assignment, gives the weight of the OR-to-AND arc:

Definition 14 (OR-to-AND weights): Given an AND/OR graph of a belief network $\mathcal{B}$, the weight $w_{(n,m)}(X_i, x_i)$ of arc (n, m) where X_i labels n and x_i labels m, is the *combination* of all the CPTs in $B_{\mathcal{T}}(X_i)$ assigned by values along the current path to the AND node m, π_m. Formally, $w_{(n,m)}(X_i, x_i) = \otimes_{P_j \in B_{\mathcal{T}}(X_i)} P_j(asgn(\pi_m)[scope(P_j)])$.

Definition 15 (weight of a solution subtree): Given a weighted AND/OR graph of a belief network $\mathcal{B}$, and given a solution subtree t having the OR-to-AND set of arcs $arcs(t)$, the weight of t is defined by $w(t) = \otimes_{e \in arcs(t)} w(e)$.

Example 6: Figure 9 shows a weighted AND/OR tree for a belief network. Figure 9a shows the primal graph, Fig. 9b is the pseudo tree, and Fig. 9c shows the conditional probability tables. Figure 9d shows the weighted AND/OR search tree. Naturally, this tree could be transformed into the context minimal AND/OR graph, similar to the one in Fig. 8b.

Value of a node: When solving a reasoning task, each node of the AND/OR graph can be associated with a *value*. The value could be the number of solutions restricted below the node, or the probability of those solutions. Whenever a subproblem is solved, the solution value is recorded and pointed to by the context assignment of the node. Whenever the same context assignment is encountered again along a different path, the recorded solution value is retrieved.

Example 7: We refer again to the example in Fig. 9. Considering a constraint network that imposes that $D = 1$ and $E = 0$ (this can also be evidence in the belief network), the trace of the depth first search algorithm without caching (algorithm AND-OR-$_{\text{CPE}}$, described later in Section 6) is given in Fig. 10. To make the computation straightforward, the consistent leaf AND nodes are given a value of 1 (shown under the square node). The final value of each node is shown to its left, while the OR-to-AND weights are shown close to the arcs. The computation of the final value is detailed for one OR node (along the path $A = 0$, $B = 1$, C) and one AND node (along the path $A = 1$, $B = 1$).

In Sections 5 and 6 we will extend the inference and search algorithms to solve the CPE query over the new framework of mixed networks.

5 Inference algorithms for processing mixed networks

We will focus on the CPE task of computing $P(\varphi)$, where φ is the constraint expression or CNF formula, and show how we can answer the query using inference. A number of related tasks can be easily derived by changing the appropriate operator (e.g. using maximization for maximum probable explanation - MPE, or summation and maximization for maximum a posteriori hypothesis - MAP). The results in this section are based on the work in Dechter and Larkin [15] and some of the work in Larkin and Dechter [26].

5.1 Inference by bucket elimination

We will first derive a bucket elimination algorithm for mixed networks when the deterministic component is a CNF formula and latter will show how it generalizes to any constraint expression. Given a mixed network $\mathcal{M}_{(\mathcal{B},\varphi)}$, where φ is a CNF formula defined on a subset of variables Q, the *CPE* task is to compute:

$$P_{\mathcal{B}}(\phi) = \sum_{\bar{x}_Q \in models(\phi)} P(\bar{x}_Q).$$

Using the belief network product form we get:

$$P(\phi) = \sum_{\{\bar{x}|\bar{x}_Q \in models(\phi)\}} \prod_{i=1}^{n} P(x_i|x_{pa_i}).$$

We assume that X_n is one of the CNF variables, and we separate the summation over X_n and $\mathbf{X} \setminus \{X_n\}$. We denote by γ_n the set of all clauses that are defined on X_n and by β_n all the rest of the clauses. The scope of γ_n is denoted by Q_n, we define $S_n = \mathbf{X} \setminus Q_n$ and U_n is the set of all variables in the scopes of CPTs and clauses that are defined over X_n. We get:

$$P(\phi) = \sum_{\{\bar{x}_{n-1}|\bar{x}_{S_n} \in models(\beta_n)\}\{x_n|\bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{i=1}^{n} P(x_i|x_{pa_i}).$$

Denoting by t_n the set of indices of functions in the product that *do not* mention X_n and *by* $l_n = \{1, \ldots, n\} \setminus t_n$ we get:

$$P(\phi) = \sum_{\{\bar{x}_{n-1}|\bar{x}_{S_n} \in models(\beta_n)\}} \prod_{j \in t_n} P_j \cdot \sum_{\{x_n|\bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{j \in l_n} P_j.$$

Therefore:

$$P(\phi) = \sum_{\{\bar{x}_{n-1}|\bar{x}_{S_n} \in models(\beta_n)\}} \left(\prod_{j \in t_n} P_j \right) \cdot \lambda^{X_n},$$

where λ^{X_n} is defined over $U_n - \{X_n\}$, by

$$\lambda^{X_n} = \sum_{\{x_n|\bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{j \in l_n} P_j. \quad (1)$$

The case of observed variables: When X_n is observed, or constrained by a literal, the summation operation reduces to assigning the observed value to each of its CPTs *and* to each of the relevant clauses. In this case Eq. 1 becomes (assume $X_n = x_n$ and $P_{=x_n}$ is the function instantiated by assigning x_n to X_n):

$$\lambda^{x_n} = \prod_{j \in l_n} P_{j=x_n}, \quad if \bar{x}_{Q_n} \in m(\gamma_n \wedge (X_n = x_n)). \tag{2}$$

Otherwise, $\lambda^{x_n} = 0$. Since $\bar{x}_{Q_n}$ satisfies $\gamma_n \wedge (X_n = x_n)$ only if $\bar{x}_{Q_n - X_n}$ satisfies $\gamma^{x_n} = resolve(\gamma_n, (X_n = x_n))$, we get:

$$\lambda^{x_n} = \prod_{j \in l_n} P_{j=x_n} \quad if \bar{x}_{Q_n - X_n} \in m(\gamma_n^{x_n}). \tag{3}$$

Therefore, we can extend the case of observed variable in a natural way: CPTs are assigned the observed value as usual while clauses are individually resolved with the unit clause ($X_n = x_n$), and both are moved to appropriate lower buckets.

In general, when we don't have evidence in the bucket of X_n we should compute λ^{X_n}. We need to collect all CPTs and clauses mentioning X_n and then compute the function in Eq. 1. The computation of the rest of the expression proceeds with X_{n-1} in the same manner. This yields algorithm *Elim-CPE* described in Algorithm 2 with Procedure `Process-bucket`$_p$. The elimination operation is denoted by the general operator symbol ⇓ that instantiates to summation for the current query.

Algorithm 2: ELIM-CPE

input : A belief network $\mathcal{B} = \{P_1, \ldots, P_n\}$; a CNF formula on k propositions $\varphi = \{\alpha_1, \ldots \alpha_m\}$ defined over k propositions; an ordering of the variables, $d = \{X_1, \ldots, X_n\}$.

output : The belief $P(\varphi)$.

1 Place buckets with unit clauses last in the ordering (to be processed first). `// Initialize`

Partition $\mathcal{B}$ and φ into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all the CPTs and clauses whose highest variable is X_i.

Put each observed variable into its appropriate bucket. Let $S_1, \ldots, S_j$ be the scopes of the CPTs, and $Q_1, \ldots Q_r$ be the scopes of the clauses. (We denote probabilistic functions by λs and clauses by αs).

2 **for** $p \leftarrow n$ **down to** 1 **do** `// Backward`

Let $\lambda_1, \ldots, \lambda_j$ be the functions and $\alpha_1, \ldots, \alpha_r$ be the clauses in $bucket_p$

`Process-bucket`$_p(\Sigma, (\lambda_1, \ldots, \lambda_j), (\alpha_1, \ldots, \alpha_r))$

3 **return** $P(\varphi)$ as the result of processing $bucket_1$.

Procedure `Process-bucket`$_p$ $(\Downarrow, (\lambda_1, \ldots, \lambda_j), (\alpha_1, \ldots, \alpha_r))$

if *$bucket_p$ contains evidence $X_p = x_p$ (or a unit clause)* **then**

1. Assign $X_p = x_p$ to each λ_i, and put each resulting function in the bucket of its latest variable

2. Resolve each α_i with the unit clause, put non-tautology resolvents in the buckets of their latest variable and **move any bucket with unit clause to top of processing**

else

Generate $\lambda^p = \Downarrow_{(x_p \bar{x}_{U_p} \in models(\alpha_1, \ldots, \alpha_r))} \prod_{i=1}^{j} \lambda_i$

Add λ^p to the bucket of the latest variable in U_p, where $U_p = \cup_{i=1}^{j} S_i \cup_{i=1}^{r} Q_i - \{X_p\}$

For every ordering of the propositions, once all the CPTs and clauses are partitioned (each clause and CPT is placed in the bucket of the latest variable in their scope), the algorithm process the buckets from last to first. It process each bucket as either *evidence bucket*, if we have a unit clause (evidence), or as a *function computation* bucket, otherwise. Let $\lambda_1, \ldots \lambda_t$ be the probabilistic functions in bucket P over scopes $S_1, \ldots, S_t$ and $\alpha_1, \ldots \alpha_r$ be the clauses over scopes $Q_1, \ldots, Q_r$. The algorithm computes a new function λ^P over $U_p = S \cup Q - \{X_p\}$ where $S = \cup_i S_i$, and $Q = \cup_j Q_j$, defined by:

$$\lambda^P = \sum_{\{x_p | \bar{x}_Q \in models(\alpha_1, \ldots, \alpha_r)\}} \prod_j \lambda_j \tag{4}$$

From our derivation we can already conclude that:

Theorem 4 (correctness and completeness): Algorithm Elim-CPE is sound and complete for the CPE task.

Example 8: Consider the belief network in Fig. 11 and the query $\varphi = (B \vee C) \wedge (G \vee D) \wedge (\neg D \vee \neg B)$. The initial partitioning into buckets along the ordering $d = A, C, B, D, F, G$, as well as the output buckets are given in Fig. 12. We compute:

In bucket G: $\lambda^G(f, d) = \Sigma_{\{g | g \vee d = true\}} P(g|f)$

In bucket F: $\lambda^F(b, c, d) = \Sigma_f P(f|b, c)\lambda^G(f, d)$

In bucket D: $\lambda^D(a, b, c) = \Sigma_{\{d | \neg d \vee \neg b = true\}} P(d|a, b)\lambda^F(b, c, d)$

In bucket B: $\lambda^B(a, c) = \Sigma_{\{b | b \vee c = true\}} P(b|a)\lambda^D(a, b, c)\lambda^F(b, c)$

In bucket C: $\lambda^C(a) = \Sigma_c P(c|a)\lambda^B(a, c)$

In bucket A: $\lambda^A = \Sigma_a P(a)\lambda^C(a)$

The result is $P(\varphi) = \lambda^A$.

For example $\lambda^G(f, d = 0) = P(g = 1|f)$, because if $d = 0$ g must get the value "1", while $\lambda^G(f, d = 1) = P(g = 0|f) + P(g = 1|f)$.

Note that some saving due to constraints can be obtained in each function computation. Consider the bucket D that has functions over 4 variables. Brute force computation would require enumerating 16 tuples, because the algorithm has to look at all possible assignments of four binary variables. However since the processing should be restricted to tuples where b and d cannot both be true, there is a potential for restricting the computation to 12 tuples only. We will elaborate on this more later when discussing sparse function representations.

We can exploit constraints in Elim-CPE in two ways following the two cases for processing a bucket either as evidence-bucket, or as a function-computation bucket.

Exploiting constraints in evidence bucket: Algorithm Elim-CPE is already explicit in how it takes advantage of the constraints when processing an evidence bucket. It includes a unit resolution step whenever possible (see Procedure `Process-bucket`$_p$) and a dynamic reordering of the buckets that prefers processing buckets that include unit clauses. These two steps amount to applying *unit propagation* which is known to be a very effective constraint

propagation algorithm for processing CNF formulas. This may have a significant computational impact because evidence buckets are easy to process, because unit propagation increases the number of buckets that will have evidence and because treating unit clauses as evidence avoids the creation new dependencies. To further illustrate the possible impact of inferring unit clauses we look at the following example.

Example 9: Let's extend the example by adding $\neg G$ to our earlier query. This will place $\neg G$ in the bucket of G. When processing bucket G, unit resolution creates the unit clause D, which is then placed in bucket D. Next, processing bucket F creates a probabilistic function on the two variables B and C. Processing bucket D that now contains a unit clause will assign the value D to the CPT in that bucket and apply unit resolution, generating the unit clause $\neg B$ that is placed in bucket B. Subsequently, in bucket B we can apply unit resolution again, generating C placed in bucket C, and so on. In other words, aside from bucket F, we were able to process all buckets as observed buckets, by propagating the observations. (See Fig. 13.) To incorporate dynamic variable ordering, after processing bucket G, we move bucket D to the top of the processing list (since it has a unit clause). Then, following its processing, we process bucket B and then bucket C, then F, and finally A.

Exploiting constraints in function computation: Sometimes there is substantial determinism present in a network that cannot yield a significant amount of unit clauses or shrink the domains of variables. For example, consider the case when the network is completely connected with equality constraints. Any domain value for any single variable is feasible, but there are still only k solutions, where k is the domain size. We can still exploit such constraints in the function-computation. To facilitate this we may need to consider different data structures, other than tables, to represent the CPT functions.

In Larkin and Dechter[26] we focused on this aspect of exploiting constraints. We presented the bucket-elimination algorithm called *Elim-Sparse* for the CPE query, that uses a sparse representation of the CPT functions as a relation. Specifically, instead of recording a table as large as the product of the domain sizes of all the variables, a function is maintained as a relation of non-zero probability tuples. In the above example, with the equality constraints, defining the function as a table would require a table of size k^n where n is the number of variables in the scope of the function, but only nk (k tuples of size n each) as a relation. Efficient operations to work with these functions are also available. These are mainly based on the Hash-Join procedure which is well-known in database theory [25] as described in Larkin and Dechter [26].

In Elim-Sparse, the constraints are absorbed into the (relation-based) CPTs (e.g., in a generalized arc-consistency manner) and then relational operators can be applied. Alternatively, one can also devise efficient function-computation procedures using constraint-based search schemes. We will assume the sparse function representation explicitly in the constraint-based CPE algorithm *Elim-ConsPE(i)* described in Section 5.2.2.

5.2 Extensions of Elim-CPE

Unit propagation and any higher level of constraint processing can also be applied a priori on the CNF formula before we apply Elim-CPE. This can yield stronger CNF expressions in each bucket with more unit clauses. This can also improve the function computation in non-evidence buckets. Elim-CPE(i) is discussed next.

5.2.1 Elim-CPE(i)—One form of constraint propagation is bounded resolution [43]. It applies pair-wise resolution to any two clauses in the CNF theory iff the resolvent size does not exceed a bounding parameter, i. Bounded-resolution algorithms can be applied until quiescence or in

a directional manner, called *BDR*(*i*). After partitioning the clauses into ordered buckets, each one is processed by resolution relative to the bucket's variable, with bound *i*.

This suggests extending Elim-CPE into a parameterized family of algorithms Elim-CPE(*i*) that incorporates *BDR*(*i*). All we need is to include Procedure BDR (*i*) described below in the "else" branch of the Procedure $\texttt{Process-bucket}_p$.

Procedure BDR (*i*)

if *the bucket does not have an observed variable* **then**

  for *each pair* $\{(\alpha \vee Q_j),(\beta \vee \neg Q_j)\} \subseteq bucket_p$ **do**

    if *the resolvent* $\gamma = \alpha \cup \beta$ *contains no more than i propositions* **then**

      place the resolvent in the bucket of its latest variable

5.2.2 Probability of relational constraints—When the variables in the belief network are multi-valued, the deterministic query can be expressed using a constraint expression with relational operators. The set of solutions of a constraint network can be expressed using the join operator. The join of two relations R_{AB} and R_{BC} denoted $R_{AB} \bowtie R_{BC}$ is the largest set of solutions over *A*, *B*, *C* satisfying the two constraints R_{AB} and R_{BC}. The set of solutions of the constraint expression $\mathcal{R} = \{R_1, \ldots R_t\}$ is $sol(\mathcal{R}) = \bowtie_{i=1}^{t} R_i$.

Given a belief network and a constraint expression $\mathcal{R}$ we may be interested in computing $P(\bar{x} \in sol(\mathcal{R}))$. A bucket-elimination algorithm for computing this task is a simple generalization of Elim-CPE, except that it uses the relational operators as expressed in Algorithm 4. Algorithm Elim-ConsPE uses the notion of arc-consistency which generalizes unit propagation and it is also parameterized to allow higher levels of directional *i*-consistency (DIC(i)) [14], generalizing *BDR*(*i*) (see step 1 of the "else" part of the *process-bucket-rel* procedure). The algorithm assumes sparse function representation and constraint-exploiting computation for the bucket-functions.

Clearly, in both Elim-CPE(*i*) and its generalized constraint-based version Elim-ConsPE(*i*), higher levels of constraint propagation may desirably infer more unit and non-unit clauses. They may also require more computation however and it is hard to assess in advance what level of *i* will be cost-effective. It is known that the complexity of *BDR*(*i*) and *DIC*(*i*) are *O*(exp(*i*)) and therefore, for small levels of *i* the computation is likely to be dominated by generating the probabilistic function rather than by *BDR*(*i*).

Moreover, whether or not we use high level of directional consistency to yield more evidence, a full level of directional consistency is achieved anyway by the function computation. In other words, the set of positive tuples generated in each bucket's function computation is identical to the set of consistent tuples that would have been generated by full directional consistency (also known as *adaptive-consistency* or *directional-consistency*) with the same set of constraints. Thus, full directional *i*-consistency is not necessary for the sake of function computation. It can still help inferring significantly more unit clauses (evidence) over the constraints, requiring a factor of 2 at the most for the processing of each bucket.

Algorithm 4: ELIM -CONS PE

input: A belief network $\mathcal{B} = \{P_1, \ldots, P_n\}$ where P_i's are assume to have a sparse representation; A constraint expression over *k* variables, $\mathcal{R} = \{R_{Q_1}, \ldots, R_{Q_t}\}$ an ordering $d = \{X_1, \ldots, X_n\}$

output: The belief $P(\mathcal{R})$.

1 Place buckets with observed variables last in *d* (to be processed first) `// Initialize`

Partition $\mathcal{C}$ and $\mathcal{R}$ into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all CPTs and constraints whose highest variable is X_i

Let $S_1, \ldots, S_j$ be the scopes of the CPTs, and $Q_1, \ldots Q_t$ be the scopes of the constraints.

We denote probabilistic functions as λ s and constraints by Rs

2 **for** $p \leftarrow n$ **down to** 1 **do** `// Backward`

Let $\lambda_1, \ldots, \lambda_j$ be the functions and $R_1, \ldots, R_r$ be the constraints in $bucket_p$

`Process-bucket-REL`$_p(\Sigma, (\lambda_1, \ldots, \lambda_j),(R_1, \ldots, R_r))$

3 **return** $P(\mathcal{R})$ as the result of processing $bucket_1$.

Procedure `Process-bucket-REL`$_p$ $(\Downarrow, (\lambda_1, \ldots, \lambda_j),(R_1, \ldots, R_r))$

if $bucket_p$ *contains evidence* $X_p = x_p$ **then**

1. Assign $X_p = x_p$ to each λ_i and put each resulting function in the bucket of its latest variable

2. Apply arc-consistency (or any constraint propagation) over the constraints in the bucket.

Put the resulting constraints in the buckets of their latest variable and **move any bucket with single domain to top of processing**

else

1. Apply directional i-consistency (DIC(i))

2. Generate $\lambda^p = \sum_{\{x_p | \bar{x}_{U_p} \in \bowtie_j R_j\}} \prod_{i=1}^{j} \lambda_i$ with specialized sparse operations or search-based methods.

Add λ^p to the bucket of the latest variable in U_p, where $U_p = \cup_{i=1}^{j} S_i \cup_{i=1}^{r} Q_i - \{X_p\}$

5.3 Complexity

As usual, the worst-case complexity of bucket elimination algorithms is related to the number of variables appearing in each bucket, both in the scopes of probability functions as well as in the scopes of constraints [13]. The worst-case complexity is time and space exponential in the maximal number of variables in a bucket, which is captured by the induced-width of the relevant graph. Therefore, the complexity of Elim-CPE and Elim-ConsPE is $O(r \cdot exp(w^*))$, where w^* is the induced width of the moral mixed ordered graph and r is the total number of functions [23]. In Fig. 14 we see that while the induced width of the moral graph of the belief network is just 2 (Fig. 14a), the induced width of the mixed graph of our example is 3 (Fig. 14b).

We can refine the above analysis to capture the role of constraints in generating unit clauses by constraint propagation. We can also try to capture the power of constraint-based pruning obtained in function computation. To capture the simplification associated with observed variables, we will use the notion of an *adjusted induced graph.* The adjusted induced graph is created by processing the variables from last to first in the given ordering and connecting the parents of each non-observed variables, only. The adjusted induced width is the width of the adjusted induced-graph. Figure 14c shows the adjusted induced-graph relative to the evidence $\neg G$. We see that the induced width, adjusted for this observation, is just 2 (Fig. 14c). Notice that adjusted induced-width can be computed once we observe the evidence set obtained as a result of our propagation algorithm. In summary:

Theorem 5 [15]: Given a mixed network, $\mathcal{M}$, of a belief network over n variables, a constraint expression and an ordering o, algorithm Elim-CPE is time and space $O(n \cdot exp(w^*_{\mathcal{M}}(o)))$, where $w^*_{\mathcal{M}}(o)$ is the width along o of the adjusted moral mixed induced graph.

Capturing in our analysis the efficiency obtained when exploiting constraints in function-computation is harder. The overall complexity depends on the amount of determinism in the problem. If enough is present to yield small relational CPTs, it can be fairly efficient, but if not, the overhead of manipulating nearly full tuple lists can be larger than when dealing with a table. Other structured function representations, such as decision trees [7] or rule-based systems [39] might also be appropriate for sparse representation of the CPTs.

6 AND/OR search algorithms for mixed networks

Proposition 2 ensures the equivalence of mixed networks defined by one belief network and by different constraint networks that are equivalent (i.e., that have the same set of solutions). In particular, this implies that we can process the deterministic information separately (e.g., by enforcing some consistency level, which results in a tighter representation), without losing any solution. Conditioning algorithms (search) offer a natural approach for exploiting constraints. The intuitive idea is to search in the space of partial variable assignments, and use the wide range of readily available constraint processing techniques to limit the actual traversed space. We will describe the basic principles in the context of AND/OR search spaces [17]. We will first describe the AND-OR-CPE Algorithm. Then, we will discuss how to incorporate in AND-OR-CPE techniques exploiting determinism, such as: (1) constraint propagation (look-ahead), (2) backjumping and (3) good and nogood learning.

6.1 AND-OR-CPE algorithm

Algorithm 5, AND-OR-CPE, presents the basic depth-first traversal of the AND/OR search tree (or graph, if caching is used) for solving the CPE task over a mixed network. The algorithm is similar to the one presented in Dechter and Mateescu [17]. The input is a mixed network, a pseudo tree $\mathcal{T}$ of the moral mixed graph and the context of each variable. The output is the probability that a random tuple generated from the belief network distribution satisfies the constraint query. AND-OR-CPE traverses the AND/OR search tree or graph corresponding to $\mathcal{T}$ in a DFS manner. Each node maintains a value v which accumulates the computation resulted from its subtree. OR nodes accumulate the summation of the product between each child's value and its OR-to-AND weight, while AND nodes accumulate the product of their children's values. For more information see Dechter and Mateescu [17].

Procedure `ConstraintPropagation`($\mathcal{R}$, $\bar{x}_i$)

input: A constraint network $\mathcal{R}$ = $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$; a partial assignment path $\bar{x}_i$ to variable X_i.

output: reduced domain D_i of X_i; reduced domains of future variables; newly inferred constraints.

This is a generic procedure that performs the desired level of constraint propagation, for example forward checking, unit propagation, arc consistency over the constraint network $\mathcal{R}$ and conditioned on $\bar{x}_i$.

return *reduced domain of* X_i

Example 10—We refer back to the example in Fig. 9. Consider a constraint network that is defined by the CNF formula $\varphi = (A \vee C) \wedge (B \vee \neg E) \wedge (B \vee D)$. The trace of algorithm AND-OR-CPE without caching is given in Fig. 15. Notice that the clause $(A \vee C)$ is not satisfied if $A = 0$ and $C = 0$, therefore the paths that contain this assignment cannot be part of a solution of the mixed network. The value of each node is shown to its left (the leaf nodes assume a dummy

value of 1, not shown in the figure). The value of the root node is the probability of φ. Figure 15 is similar to Fig. 10. In Fig. 10 the evidence can be modeled as the CNF formula with unit clauses $D \wedge \neg E$.

The following theorems are implied immediately from the general properties of AND/OR search algorithms [17].

Theorem 6: Algorithm AND-OR-$_{CPE}$ is sound and exact for the CPE task.

Theorem 7: Given a mixed network $\mathcal{M}$ with n variables having domain sizes bounded by k and a pseudo tree $\mathcal{T}$ of depth m of its moral mixed graph, the time complexity of AND-OR-$_{CPE}$ with no caching is $O(n \cdot k^m)$, while the space required is linear. A mixed network of treewidth w* has an AND/OR search tree whose size is $O(\exp(w^* \cdot \log n))$.

6.2 Constraint propagation in AND-OR-$_{CPE}$

As we already observed, Proposition 2 provides an important justification for using mixed networks as opposed to auxiliary networks. The constraint portion can be processed by a wide range of constraint processing techniques, both statically before search or dynamically during search [14].

We discuss here the use of constraint propagation during search, also known as look-ahead. This is a well known idea used in any constraint or SAT solver. In general, constraint propagation helps to discover (using limited computation) what variable and what value to instantiate next. The incorporation of these methods on top of AND/OR search is straightforward. For illustration, we will only consider a static variable ordering, based on a pseudo tree.

In Algorithm AND-OR-$_{CPE}$, line 11 contains a call to the generic `ConstraintPropagation` procedure consulting only the constraint subnetwork $\mathcal{R}$, conditioned on the current partial assignment. The constraint propagation is relative to the current set of constraints, the given path that defines the current partial assignment, and the newly inferred constraints, if any, that were learned during the search. Using a polynomial time algorithm, `ConstraintPropagation` may discover some variable values that cannot be extended to a full solution. These values in the domain of a variables are marked as inconsistent and can be removed from the current domain of the variable. All the remaining values are returned by the procedure as good candidates to extend the search frontier. Of course, not all the values returned by `ConstraintPropagation` are guaranteed to lead to a solution.

We therefore have the freedom to employ any procedure for checking the consistency of the constraints of the mixed network. The simplest case is when no constraint propagation is used, and only the initial constraints of $\mathcal{R}$ are checked for consistency, and we denote this algorithm by AO-C.

In the empirical evaluation, we used two forms of constraint propagation on top of AO-C. The first, yielding algorithm AO-FC, is based on *forward checking*, which is one of the weakest forms of propagation. It propagates the effect of a value selection to each future uninstantiated variable separately, and checks consistency against the constraints whose scope would become fully instantiated by just one such future variable.

The second algorithm we used is called AO-RFC, and performs a variant of *relational forward checking*. Rather than checking only constraints whose scope becomes fully assigned, AO-RFC checks all the existing constraints by looking at their projection on the current path. If the

projection is empty an inconsistency is detected. AO-RFC is computationally more expensive than AO-FC, but its search space is smaller.

SAT solvers—One possibility that was explored with success (e.g., Allen and Darwiche [1]) is to delegate the constraint processing to a separate off-the-shelf SAT solver. In this case, for each new variable assignment the constraint portion is packed and fed into the SAT solver. If no solution is reported, then that value is a dead-end. If a solution is found by the SAT solver, then the AND/OR search continues (remember that for some tasks we may have to traverse all the solutions of the graphical model, so the one solution found by the SAT solver does not finish the task). The worst-case complexity of this level of constraint processing, at each node, is exponential.

The popular variant of *unit propagation* that was exploited in Elim-CPE can be effective here too. This can also be implemented by the unit resolution engine of an available SAT solver. Such hybrid use of search and a specialized efficient SAT (or constraint) solver can be very useful, and it underlines further the power that the mixed network representation has in delimiting the constraint portion from the belief network.

Example 11—Figure 16a shows the belief part of a mixed network, and Fig. 16b the constraint part. All variables have the same domain, {1,2,3,4}, and the constraints express “less than” relations. Figure 16c shows the search space of AO-C. Figure 16d shows the space traversed by AO-FC. Figure 16e shows the space when consistency is enforced with Maintaining Arc Consistency (which enforces full arc-consistency after each new instantiation of a variable).

6.3 Backjumping

Backjumping algorithms [4,14,20,41] are backtracking search algorithms applied to the OR space, which uses the problem structure to jump back from a dead-end as far back as possible. They have been known for a long time in the constraint processing community. For probabilistic models, backjumping is very useful in the context of determinism.

In *graph-based backjumping* (GBJ) each variable maintains a graph-based induced ancestor set which ensures that no solutions are missed by jumping back to its deepest variable. If the ordering of the OR space is a DFS ordering of the primal graph, it is known [14] that all the backjumps are from a variable to its DFS parent. In Mateescu and Dechter [27] it was shown that this means that a simple AND/OR search automatically incorporates graph-based backjumping, when the pseudo tree is a DFS tree of the primal graph.

When the pseudo tree is not a DFS tree of the primal graph, it may happen that the parent of a node in the pseudo tree is not the node where graph-based backjumping would retreat in the case of OR search. An example is provided in Fig. 17. Figure 17a shows a graphical model, Fig. 17b a pseudo tree and 17c a chain driving the OR search (top down). If a dead-end is encountered at variable 3, graph-based backjumping retreats to 8 (see Fig. 17c), while simple AND/OR would retreat to 1, the pseudo tree parent. When the dead-end is encountered at 2, both algorithms backtrack to 3 and then to 1. Therefore, in such cases, augmenting AND/OR with a graph-based backjumping mechanism can provide some improvement.

We want to emphasize that the graph-based backjumping behavior is in most cases intrinsic to AND/OR search. The more advanced and computationally intensive forms of conflict directed backjumping [14,41] are not captured by the AND/OR graph, and can be implemented on top of it by analyzing the constraint portion only.

6.4 Good and nogood learning

When a search algorithm encounters a dead-end, it can use different techniques to identify the ancestor variable assignments that caused the dead-end, called a conflict-set. It is conceivable that the same assignment of that set of ancestor variables may be encountered in the future, and they would lead to the same dead-end. Rather than rediscovering it again, if memory allows, it is useful to record the dead-end conflict-set as a new constraint (or clause) over the ancestor set that is responsible for it. Recording dead-end conflict-sets is sometimes called nogood learning.

One form of nogood learning is graph-based, and it uses the same technique as graph-based backjumping to identify the ancestor variables that generate the nogood. The information on conflicts is generated from the primal graph information alone. Similar to the case of backjumping, it is easy to see that AND/OR search already provides this information in the context of the nodes. Therefore saving the information about the nogoods encountered amounts to graph-based nogood learning in the case of OR search.

If deeper types of nogood learning are desirable, they need to be implemented on top of the AND/OR search. In such a case, a smaller set than the context of a node may be identified as a culprit assignment, and may help discover future dead-ends much earlier than when context-based caching alone is used. Needless to say, deep learning is computationally more expensive and it can be facilitated via a focus on the constraint portion of the mixed network.

Traversing the context-based AND/OR graph can be understood as learning (and saving) not only the nogoods but also their logical counterparts, the *goods* (namely the consistent assignments). This is a feature that was proposed in recent years by several schemes [10,17, 44]. This is in fact the well known technique of caching that became appealing recently due to the availability of computer memory, when the task to be solved requires the enumeration of many solutions. The idea is to store the value of a solved conditioned subproblem, associating it with a minimal set of ancestor assignments that are guaranteed to root the same conditioned subproblem, and retrieve that value whenever the same set of ancestor assignments is encountered again during search.

7 Empirical evaluation

In this section we present an empirical evaluation of the inference and AND/OR search methods discussed in the previous sections.

Exploiting determinism in BE vs. search—We do not advocate here that one type of algorithms is better than the other. In fact, as an extension of the results in Mateescu and Dechter [27], it can be shown that search and inference are in general incomparable, when both are equipped with determinism exploiting tools. Like AND/OR search, bucket-elimination that uses sparse function representation can be shown to also traverse the context minimal AND/OR graph, but in a different direction and assuming a different control strategy. Inference is bottom up and breadth first, while AND/OR search is top down and depth first. As a result, we can imagine mixed networks where the determinism reveals itself close to the root of the pseudo tree, making the job of AND/OR search easier, while Bucket Elimination has to traverse all the layers bottom up, only to discover that most of the messages it has processed contain invalid tuples. Another example could show the opposite: if the determinism is closer to the leaves of the pseudo tree, then the performance of the two methods is reversed.

We should emphasize again that in making this claim about inference we assume that the CPTs use a sparse representation. In practice the impact of determinism would be manifested in generating tight functions that are sent from one bucket to another. If we consider the case

discussed after Example 8, a full table representation for four binary variables would contain 16 tuples, but a restriction to only the valid tuples might be a relational representation for only 12 of them.

Because, as explained, inference and search are in general incomparable, we will offer an experimental evaluation of each type separately, evaluating the advantages of expressing and exploiting the constraint portion separately as part of the mixed network framework.

7.1 Inference algorithms

We compared empirically five algorithms: (1) Elim-CPE (which is the same as Elim-CPE(0), which does no constraint propagation except for unit propagation); (2) Elim-CPE(i); (3) Elim-CPE-D (which derives CNF clauses from mixed CPTs and then applies Elim-CPE); (4) Elim-Hidden (this algorithm expresses each clause as a CPT with a new hidden variable, adds evidence to the hidden nodes and performs the variable elimination algorithm). All these algorithms assume that the CPTs are implemented as tables, with no sparse representation. The fifth algorithm we tested is Elim-Sparse that uses a sparse relational representation of the CPTs. We tested the algorithms on some random networks, as well as realistic networks: Insurance, Water, Mildew, Hailfinder, Munin1 and Diabetes. All algorithms use min-degree order, computed by repeatedly removing the node with the lowest degree from the graph and connecting all its neighbors. For more information see Dechter and Larkin [15].

The generator of random networks that we used is divided in two parts. The first creates a random belief network using a tuple $< n, f, d >$ as a parameter, where n is the number of variables, f is the maximum family size, and d is the fraction of deterministic entries in CPTs. Parents are chosen at random from the preceding variables in a fixed ordering. The entries of the CPTs are filled in randomly. The second part generates a 3-CNF query using a pair of parameters $< c, e >$ where c is the number of 3-CNF clauses (clauses are randomly chosen and each is given a random truth value) and e is the number of observations.

We first show a comparison of Elim-CPE-D and Elim-CPE on some random networks, in Fig. 18. As mentioned before, the difference between the two algorithms is that Elim-CPE-D extracts deterministic information from CPTs. The figure shows a scatter plot of running times measured in seconds. The results show that extracting deterministic information is beneficial on these instances.

We tested the algorithms on the Insurance network, which is a realistic network for evaluating car insurance risks that contains deterministic information. It has 27 variables. In the experiments reported in Table 1, Elim-CPE-D outperformed Elim-CPE substantially. Figure 19 contrasts Elim-CPE with Elim-Hidden on the Insurance network.

We also tried the Hailfinder network, another benchmark that has 56 variables and includes deterministic information. It is a normative system that forecasts severe summer hail in northeast Colorado. The results are reported in Table 2. Here again the results are consistent with earlier observations that Elim-CPE-D was the most efficient. In both of these networks we have determinism created by the network and the query.

We also present here some results that include the algorithm Elim-Sparse [26], where the CPTs are represented in relational form, by storing only the valid tuples. Table 3 shows a comparison of Elim-Bel (EB), Elim-CPE (EC) and Elim-Sparse (ES). We generated 50 random queries for each network, testing the three algorithms on each. The last two columns show the ratio of times of Elim-Bel to Elim-Sparse (B/S), and of Elim-CPE to Elim-Sparse (C/S). Elim-Sparse was considerably faster than Elim-CPE on Insurance, Water, Mildew, and Munin1 (by a factor of 4 or more), but less so on Hailfinder and Diabetes (less than twice as fast). In general Elim-

Sparse is more efficient than Elim-CPE especially with increasing determinism. However, the high constant factor due to manipulation of tuple lists may prove to be too big an overhead for low determinism.

7.2 AND/OR search algorithms

We provide here an evaluation of AND/OR search algorithms for mixed networks. We ran our algorithms on mixed networks generated randomly uniformly given a number of input parameters: *N*—number of variables; *K*—number of values per variable; *R*—number of root nodes for the belief network; *P*—number of parents for a CPT; *C*—number of constraints; *S*—the scope size of the constraints; *t*—the tightness (percentage of the allowed tuples per constraint). (N,K,R,P) defines the belief network and (N,K,C,S,t) defines the constraint network. We report the time in seconds, number of nodes expanded and number of dead-ends encountered (in thousands), and the number of consistent tuples of the mixed network (*#sol*). In tables, w^* is the induced width and *h* is the height of the pseudo tree.

We compared four algorithms: 1) AND-OR-$_{CPE}$, denoted here AO-C; 2) AO-FC and 3) AO-RFC (described in previous section); 4) BE—bucket elimination (which is equivalent to Elim-Bel) on the auxiliary network; the version we used in BE is the basic one for belief networks, without any constraint propagation and any constraint testing, namely we did not use the Elim-cpe type algorithms that exploit determinism. We tried different levels of caching for the AND/OR algorithms, denoted in the tables by *i* (i-bound, this is the maximum scope—— size of the tables that are stored). $i = 0$ stands for linear space search. Caching is implemented based on context as described in Section 6.

Tables 4, 5, and 6 show a comparison of the linear space and caching algorithms exploring the AND/OR space with varying levels of constraint propagation. We ran a large number of cases and this is a typical sample. Notice that the domain size is increased to $K = 3$.

Table 4 shows a medium sized mixed network, across the full range of tightness for the constraint network. For linear space ($i = 0$), we see that more constraint propagation helps for tighter networks ($t = 20$), AO-RFC being faster than AO-FC. As the constraint network becomes loose, the effort of AO-RFC does not pay off anymore. When almost all tuples become consistent, any form of constraint propagation is not cost effective, AO-C being the best choice in such cases ($t = 80, 100$). For each type of algorithm, caching improves the performance. We can see the general trend given by the bold figures.

Table 5 shows results for large mixed networks ($w^* = 28, 41$). These problems have an inconsistent constraint portion ($t = 10, 20, 30$). AO-C was much slower in this case, so we only include results for AO-FC and AO-RFC. For the smaller network ($w^* = 28$), AO-RFC is only slightly better than AO-FC. For the larger one ($w^* = 41$), we see that more propagation helps. Caching doesn't improve either of the algorithms here. This means that for these inconsistent problems, constraint propagation is able to detect many of the nogoods easily, so the overhead of caching cancels out its benefits (only nogoods can be cached for inconsistent problems). Note that these problems are infeasible for brute-force BE that does not include constraint propagation, due to high induced width. They may still be feasible for Elim-CPE(i) or Elim-Sparse though.

Table 6 shows a comparison between search algorithms and brute-force BE. All instances for $t < 40$ were inconsistent and the AO algorithms were much faster than BE, even with linear space. Between $t = 40 - 60$ we see that BE becomes more efficient than AO, and may be comparable only if AO is given the same amount of space as BE.

There is an expected trend with respect to the size of the traversed space and the dead-ends encountered. We see that the more advanced the constraint propagation technique, the less nodes the algorithm expands, and the less dead-ends it encounters. More caching also has a similar effect.

7.3 AND/OR solution counting

We present here results on pure constraint networks, for the task of solution counting. While this may seem to bias our mixed representation to an extreme, the results are in fact very relevant for processing mixed networks. The amount of computation (number of nodes explored in the AND/OR space) is the same as in the case where a belief network would exist on top of the constraint network. The only missing part here is the computation of probabilities (or weights) corresponding to partial assignments. Instead, we compute a count of the solutions. These results also show a comparison of AND/OR search with the traditional type of OR search that does not exploit problem structure but follows a chain pseudo tree.

Tables 7 and 8 show an ample comparison of the algorithms on moderate size problems which allowed bucket elimination to run. The bolded time numbers show the best values in each column. The most important thing to note is the vast superiority of AND/OR space over the traditional OR space. Only for the very tight problems (t = 10%−40%), which are also inconsistent, the two search spaces seem to be comparable. The picture is clearer if we look at the number of expanded nodes and number of dead-ends. When the problems are loose and have a large number of solutions AND/OR algorithms are orders of magnitudes better (see *#n*, *#d* bolded figures for *i=9* in Table 7, and for *i=13* in Table 8, where A/O FC explores a space two orders of magnitude smaller than that of OR FC, resulting in a time two orders of magnitude smaller). In Table 7 we also see the impact of more constraint propagation. The RFC algorithms always explore a smaller space than the FC, but this comes with an overhead cost, and may not always be faster. For BE we only report time, which is not sensitive to the tightness of the problem, so we see that for tight networks search can be faster than BE, if BE is insensitive to determinism. Clearly, a comparison with Elim-CPE(i) or Elim-Sparse may show a different picture.

Caching doesn't seem to play a big role in this first set of problems. Especially, for inconsistent networks, caching doesn't improve performance. This is probably because the type of networks we generate turn out to be fairly easy for forward checking, so even without caching the nogoods of the inconsistent networks, forward checking is able to easily detect them.

Table 9 shows an example where caching is useful. This is again a smaller problem for which A/O FC could be run even for t = 100%. When problems become loose, caching is essential to speed up the search.

8 Related work

The idea of combining probabilistic information with deterministic relationships is fundamental, and has been explored in different communities, as we have already mentioned in the introduction and throughout the paper. In the following subsections we present the related work structured along two directions: 1) languages that combine logic and probabilities; 2) computational issues of processing mixed probabilistic and deterministic information.

8.1 Languages combining logic and probability

Combining probabilistic information and first-order logic has been a long-standing goal in AI. This problem has been under intense investigation in recent years, especially because of its relevance to statistical relational learning. Most of the early approaches to combining first-order logic and Bayesian networks focused on restricted subsets such as Horn clauses as the

basic representation [32,38,46]. As a result, one of the main limitations was the combinatorial blowup of the these models. A significant improvement was achieved in Koller and Pfeffer [24], where frame-based representation systems are combined with Bayesian networks. This approach allows frame knowledge bases to be annotated with probabilistic information, making them more suitable to real-world applications.

Markov logic networks [42] is a recent approach that combines first-order logic and probabilistic graphical models by attaching a weight to each formula of a knowledge base. Another recently introduced formalism is Bayesian logic (BLOG) [30]. BLOG is a first-order probabilistic modeling language that compactly and intuitively defines probability distributions over configurations of varying sets of objects. Its purpose is to provide a language for models that handle objects that are not known a priori.

8.2 Computational aspects

When processing Bayesian networks that contain determinism (namely, CPTs with zero probability tuples), an important aspect is the encoding of the determinism in the function representation. As we described earlier in the paper, if a lot of determinism is present, it may be beneficial to represent the functions in relational form as lists of valid tuples [26]. Other structured function representations, such as decision trees [7] or rule-based systems [39] are also possible, as we noted earlier.

Recursive conditioning (RC) [10] is an algorithm that exploits the problem structure and traverses an AND/OR search space. In Allen and Darwiche [1], RC was extended with unit resolution (based on the zChaff SAT solver [31]) to effectively deal with determinism in Bayesian networks, especially for the domain of genetic linkage analysis. In certain cases, this results in significant reduction of the solving time. As we have already mentioned, any SAT or constraint solver can be employed to process the deterministic information.

Another algorithm similar to AND/OR and RC is Value Elimination [2]. The key property of Value Elimination is the ability to handle dynamic variable orderings and caching simultaneously, while maintaining in principle the same worst case complexity (i.e., exponential in the treewidth). This is realized however through the use of hash tables, and some constant access assumptions are necessary. The work of Sang et al. [44] combines component caching (essentially formula caching in SAT) with clause learning and shows that on many instances it improves over existing algorithms for #SAT by orders of magnitude.

The presence of deterministic information hidden within a probabilistic model also inspired the idea of finding triangulations (or variable orderings) that correspond to minimal computation. Therefore, besides the structural information of the primal graph, the determinism can reveal that the inconsistent assignments do not need to be enumerated in order to process the probabilistic information. The work of Bartels and Bilmes [3] shows that large-clique triangulations can sometimes lead to smaller computational effort when processing stochastic/deterministic graphical models. A more recent investigation of the search space size in the presence of determinism appears in Otten and Dechter [35].

The mixed network framework can facilitate compilation algorithms, that transform a graphical model into a single data structure that can capture compactly probabilistic and deterministic information. These include arithmetic circuits [8,11], probabilistic decision graphs [22] and AND/OR weighted decision diagrams [28]. The mixed network that we introduced in this paper can be viewed as a unifying framework within which all the above mentioned approaches can be studied and compared.

9 Discussion and conclusion

We presented the framework of *mixed networks* that combines belief and constraint networks. One primary benefit of this framework is semantic clarity. This feature is essential in modeling real life applications, an issue that we only touched upon in this paper via the motivating examples. In particular we can view a belief network having a set of variables instantiated (e.g., evidence) as a mixed network, by regarding the evidence set as a set of constraints. The dm-separation which we presented extends the d-separation of pure belief networks to the mixed network in a natural way, and provides a criterion for characterizing the notion of minimal I-mapness. Proposition 2, which defines the equivalence of mixed networks, gives blessing for processing the deterministic information separately by constraint propagation methods, rather than incorporating it in probability tables.

The second principal benefit of mixed networks is computational. The mixed networks invite the exploitation of probabilistic and deterministic information building upon their respective strengths. Indeed, our theoretical and empirical analysis showed how computation can be improved both within variable elimination and search and demonstrated the impact of constraint processing within each of these reasoning schemes.

Lets discuss further the ability of variable elimination compared with search in exploiting constraints alongside the probabilistic functions. It is often believed that search schemes can be more effective in accommodating constraints than variable elimination. Indeed, if the CPTs are expressed as full tables and if we have a problem having a significant amount of determinism, inference-based schemes can be far less effective. On the other hand if the problem has very little determinism (i.e., the CPTs are nearly positive and the constraint portion is very loose) brute-force table-based bucket elimination is likely to be far more efficient than search, assuming enough memory is available. Both of these cases were demonstrated empirically when we compared the brute-force BE algorithm with constraint-exploiting AND/OR search (Section 7) on tight and loose problems. If however the CPTs are expressed in a sparse manner, and accompanied with efficient processing algorithms, then in the presence of determinism variable elimination (i.e., Elim-Sparse, or even Elim-CPE(i)) may be more efficient than search in some of the cases. In general however they are incomparable as explained earlier.

One should note that while determinism in search is exploited by pruning the search space, determinism in variable elimination can be exploited by computing tight functions. In that case different choices of variable ordering can make one approach better than the other. For an elaborate comparison of variable elimination vs. AND/OR search see Mateescu and Dechter [27].

The relative advantages and the possible combination of the different algorithms presented here is left for future work. A wide variety of hybrid search and inference algorithms can be designed and they can also be adapted for approximate computation.

Acknowledgments

This work was supported in part by the NSF grant IIS-0713118 and by the NIH grant R01-HG004175-02.

References

1. Allen, D.; Darwiche, A. New advances in inference by recursive conditioning. Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03); 2003. p. 2-10.

2. Bacchus, F.; Dalmao, S.; Pitassi, T. Value elimination: Bayesian inference via backtracking search. Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI'03); 2003. p. 20-28.

3. Bartels, C.; Bilmes, JA. Non-minimal triangulations for mixed stochastic/deterministic graphical models. Proceedings of the Twenty Second Conference on Uncertainty in Artificial Intelligence (UAI'06); 2006.

4. Bayardo, R.; Miranker, D. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. Proceedings of the Thirteenth National Conference on Artificial Intelligence; 1996. p. 298-304.

5. Bertele, U.; Brioschi, F. Nonserial Dynamic Programming. Academic; London: 1972.

6. Bodlaender, HL.; Gilbert, JR. Tech Rep. Utrecht University; 1991. Approximating treewidth, pathwidth and minimum elimination tree-height.

7. Boutilier, C.; Friedman, N.; Goldszmidt, M.; Koller, D. Context-specific independence in Bayesian networks. Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI'96); San Francisco. 1–4 August 1996; p. 115-123.

8. Chavira M, Darwiche A, Jaeger M. Compiling relational Bayesian networks for exact inference. Int J Approx Reason 2006;42(1–2):4–20.

9. Cooper G. The computational complexity of probabistic inferences. Artif Intell 1990;42:393–405.

10. Darwiche A. Recursive conditioning. Artif Intell 2001;125(1–2):5–41.

11. Darwiche, A. A logical approach to factoring belief networks. Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR'02); Toulouse. 22–25 April 2002; p. 409-420.

12. Dechter, R. Bucket elimination: a unifying framework for probabilistic inference algorithms. Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI'96); San Francisco. 1–4 August 1996; p. 211-219.

13. Dechter R. Bucket elimination: a unifying framework for reasoning. Artif Intell 1999;113:41–85.

14. Dechter, R. Constraint Processing. Morgan Kaufmann; San Mateo: 2003.

15. Dechter, R.; Larkin, D. Hybrid processing of belief and constraints. Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01); Seattle. August 2001; p. 112-119.

16. Dechter, R.; Mateescu, R. Mixtures of deterministic-probabilistic networks and their AND/OR search space. Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04); Banff. 7–11 July 2004; p. 120-129.

17. Dechter R, Mateescu R. AND/OR search spaces for graphical models. Artif Intell 2007;171(2–3): 73–106.

18. Dechter R, Pearl J. Network-based heuristics for constraint satisfaction problems. Artif Intell 1987;34 (1):1–38.

19. Freuder, EC.; Quinn, MJ. Taking advantage of stable sets of variables in constraint satisfaction problems. Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI'85); Los Angeles. 18–23 August 1985; p. 1076-1078.

20. Gaschnig, J. Tech Rep. Carnegie Mellon University; 1979. Performance measurement and analysis of search algorithms; p. CMU-CS-79-124.

21. Heckerman, D. A tractable inference algorithm for diagnosing multiple diseases. Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI'89); 1989. p. 163-172.

22. Jaeger M. Probabilistic decision graphs—combining verification and AI techniques for probabilistic inference. Int J Uncertain Fuzziness Knowl-Based Syst 2004;12:19–42.

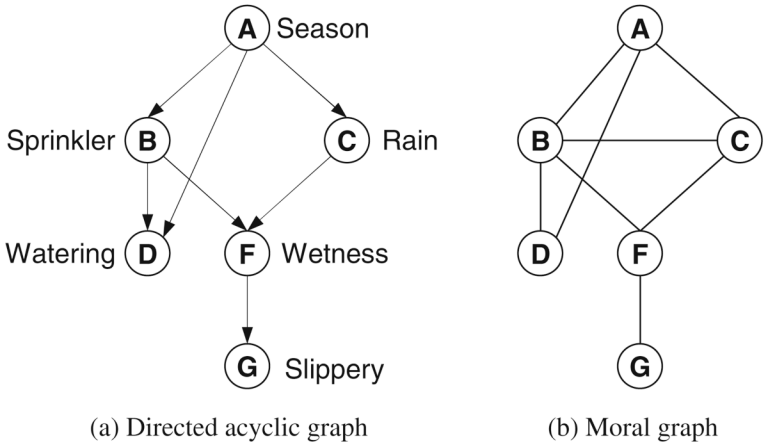
23. Kask K, Dechter R, Larrosa J, Dechter A. Unifying cluster-tree decompositions for reasoning in graphical models. Artif Intell 2005;166(1–2):165–193.

24. Koller, D.; Pfeffer, A. Probabilistic frame-based systems. Proceedings of the Fifteenth National Conference of Artificial Intelligence (AAAI'98); Madison. July 1998; p. 580-587.

25. Korth, H.; Silberschatz, A. Database System Concepts. McGraw-Hill; New York: 1991.

26. Larkin, D.; Dechter, R. Bayesian inference in the presence of determinism. The Ninth International Workshop on Artificial Intelligence and Statistics (AISTATS'03); Key West. January 2003;

27. Mateescu, R.; Dechter, R. The relationship between AND/OR search and variable elimination. Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence (UAI'05); Edinburgh. 26–29 July 2005; p. 380-387.
28. Mateescu, R.; Dechter, R. AND/OR multi-valued decision diagrams (AOMDDs) for weighted graphical models. Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence (UAI'07); Vancouver. July 2007;
29. McEliece R, MacKay D, Cheng JF. Turbo decoding as an instance of Pearl's belief propagation algorithm. IEEE J Sel Areas Commun 1998;16(2):140–152.
30. Milch, B.; Marthi, B.; Sontag, D.; Russell, S.; Ong, DL.; Kolobov, A. Blog: probabilistic models with unknown objects. Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05); Edinburgh. 30 July–5 August 2005; p. 1352-1359.
31. Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; Malik, S. Chaff: engineering an efficient SAT solver. Proceedings of the Design and Automation Conference (DAC'01); Las Vegas. June 2001; p. 530-535.
32. Ngo L, Haddawy P. Answering queries from context-sensitive probabilistic knowledge bases. Theor Comp Sci 1997;171(1–2):147–177.
33. Nilsson, NJ. Principles of Artificial Intelligence. Tioga; Palo Alto: 1980.
34. Ott, J. Analysis of Human Genetic Linkage. The Johns Hopkins University Press; Baltimore: 1999.
35. Otten, L.; Dechter, R. Bounding search space size via (hyper)tree decompositions. Proceedings of the Twenty Fourth Conference on Uncertainty in Artificial Intelligence (UAI'08); Helsinki. July 2008; p. 452-459.
36. Parker, R.; Miller, R. Using causal knowledge to create simulated patient cases: the CPCS project as an extension of INTERNIST-1. Proceedings of the Eleventh Symposium on Computer Applications in Medical Care; Washington, D.C.. November 1987; p. 473-480.
37. Pearl, J. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann; San Francisco: 1988.
38. Poole D. Probabilistic Horn abduction and Bayesian networks. Artif Intell 1993;64:81–129.
39. Poole, D. Probabilistic partial evaluation: exploiting structure in probabilistic inference. IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97); Nagoya. 23–29 August 1997; p. 1284-1291.
40. Portinale, L.; Bobbio, A. Bayesian networks for dependency analysis: an application to digital control. Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99); Stockholm. 30 July–1 August 1999; p. 551-558.
41. Prosser P. Hybrid algorithms for constraint satisfaction problems. Comput Intell 1993;9(3):268–299.
42. Richardson M, Domingos P. Markov logic networks. Mach Learn 2006;62(1–2):107–136.
43. Rish I, Dechter R. Resolution vs. search; two strategies for SAT. J Autom Reason 2000;24(1/2):225–275.
44. Sang, T.; Bacchus, F.; Beam, P.; Kautz, H.; Pitassi, T. Combining component caching and clause learning for effective model counting. Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04); Vancouver. 10–13 May 2004; p. 20-28.
45. Shenoy P. Valuation-based systems for Bayesian decision analysis. Oper Res 1992;40:463–484.
46. Wellman M, Breese J, Goldman R. From knowledge bases to decision models. Knowl Eng Rev 1992;7:35–53.
47. Zhang, N.; Poole, D. A simple approach to Bayesian network computations. Proceedings of the Tenth Canadian Conference on Artificial Intelligence; Seattle. 31 July–4 August 1994; p. 171-178.



(a) Directed acyclic graph (b) Moral graph

Fig. 1. Belief network (a, b)

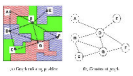


Fig. 2. Constraint network (a,b)

Fig. 3. Two-layer network with root not-equal constraints (Java Bugs)

Fig. 4. Mixed network for class scheduling

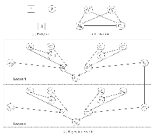


Fig. 5. Genetic linkage analysis (a–c)

Fig. 6. Example of dm-separation (a–c)

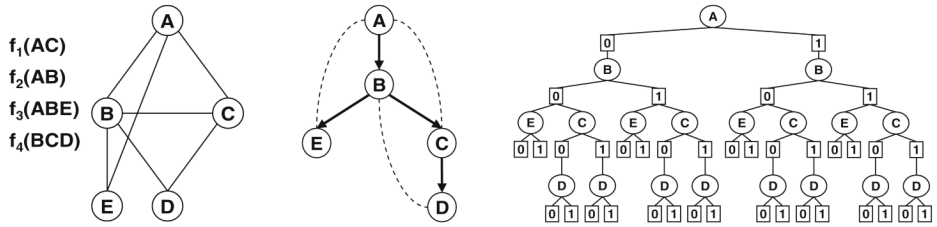
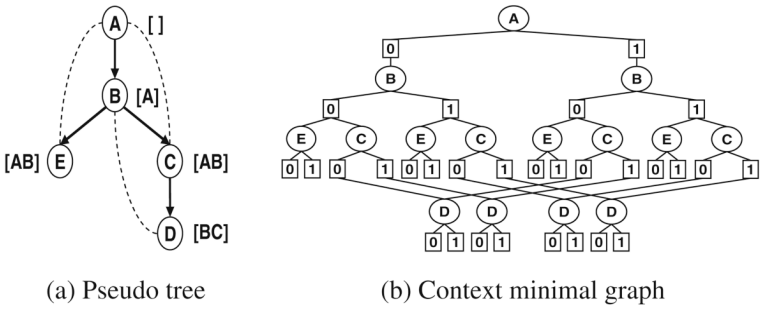


Fig. 7. AND/OR search tree (a–c)



(a) Pseudo tree (b) Context minimal graph

Fig. 8. AND/OR search graph (a, b)

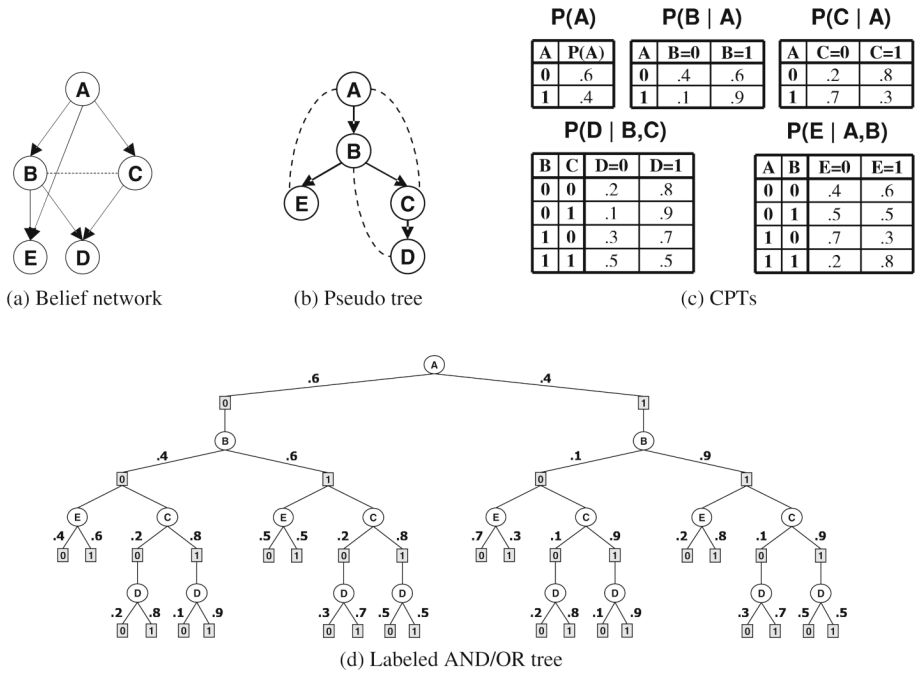


Fig. 9. Labeled AND/OR search tree for belief networks (a–c)

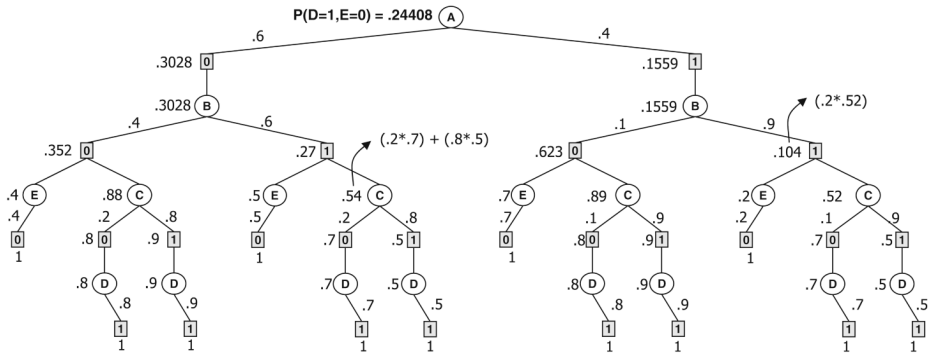
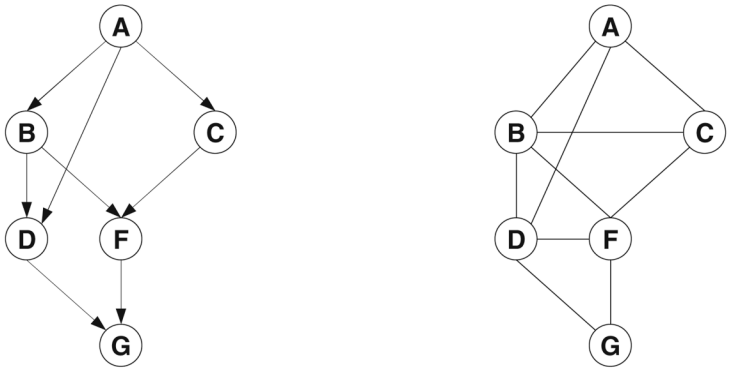


Fig. 10. AND/OR search tree with final node values



(a) Directed acyclic graph (b) Moral graph

Fig. 11. Belief network (a, b)

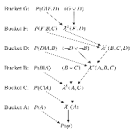


Fig. 12. Execution of ELIM-CPE

Fig. 13. Execution of ELIM-CPE (evidence ¬*G*)

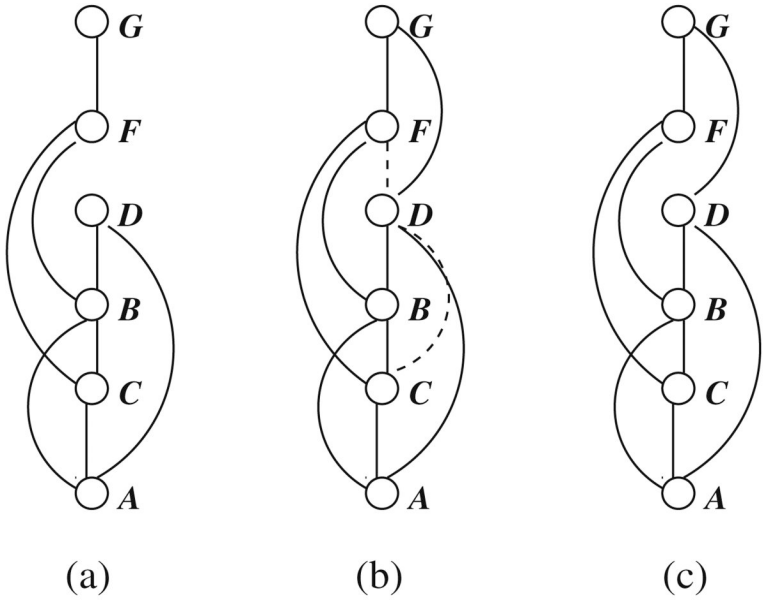


Fig. 14. Induced graphs: a moral graph; b mixed graph; c adjusted (for ¬*G*) graph

Fig. 15. Mixed network defined by the query $\varphi = (A \vee C) \wedge (B \vee \neg E) \wedge (B \vee D)$

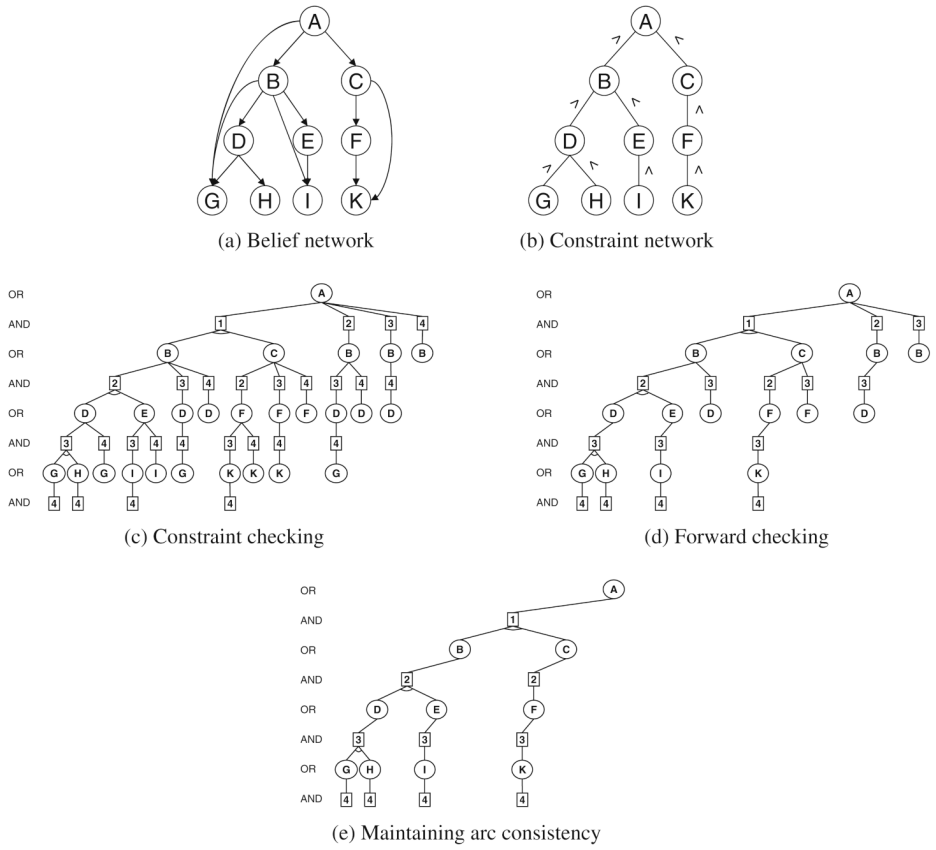


Fig. 16. Traces of AND-OR-CPE with various levels of constraint propagation (a–e)

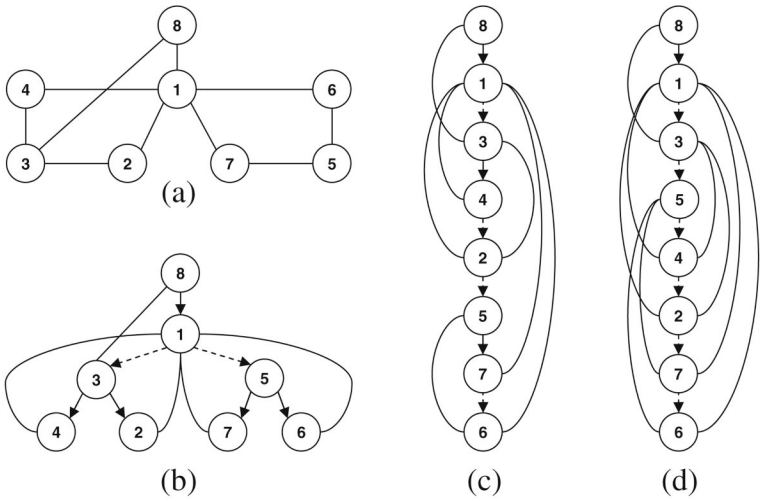


Fig. 17. Graph-based backjumping and AND/OR search (a–d)

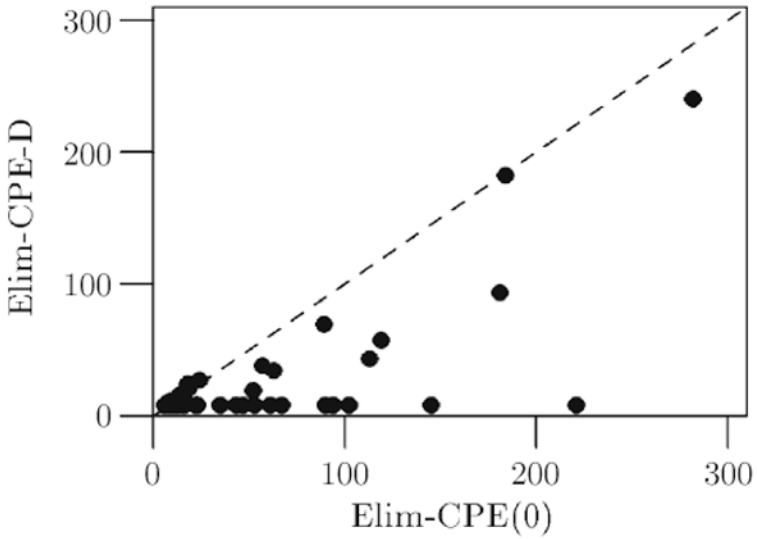


Fig. 18. Random networks; 48 instances; network parameters < 80, 4, 0.75 > and query parameters < 0, 10 >

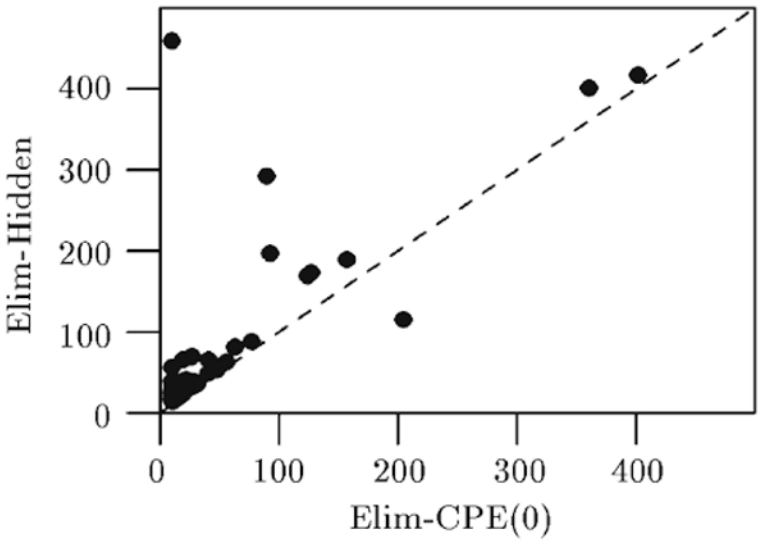


Fig. 19. Insurance network; 50 test instances; query parameters < 15, 5 >

Table 1

Insurance network (27 variables); 50 test instances; query parameters < 20, 5 >

Algorithm	Time	mf	C.	U.	F.
Elim-CPE-D:	48	8	210	1	302
Elim-CPE(15):	64	9	12	1	0
Elim-CPE(0):	61	9	6	0	0
Elim-Hidden:	104	10	0	0	0

Table 2

Hailfinder network (56 variables); 50 test instances; query parameters < 15, 15 >

Algorithm	Time	mf	C.	U.	F.
Elim-CPE-D:	4	4	269	1	501
Elim-CPE(15):	16	6	7	1	0
Elim-CPE(0):	16	6	7	1	0
Elim-Hidden:	33	7	0	0	0

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.

Table 3

Average times

Network	EB	EC	ES	B/S	C/S
Insurance	3.56	1.05	0.24	14.83	4.38
Water	4.65	3.22	0.29	16.03	11.10
Mildew	7.64	4.51	0.94	8.15	4.81
Hailfinder	1.99	1.14	0.99	2.01	1.15
Munin1	15.92	3.58	0.84	18.95	4.26
Diabetes	18.77	12.20	9.67	1.94	1.26

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.

Table 4

AND/OR search algorithms (1): random networks; induced width 12; pseudo tree depth 19; averages taken over 20 instances

N=40, K=2, R=2, P=2, C=10, S=4, 20 instances, w*=12, h=19											
t	i	Time AO-			Nodes (*1000) AO-			Dead-ends (*1000) AO-			#sol
		C	FC	RFC	C	FC	RFC	C	FC	RFC	
20	0	0.671	0.056	0.022	153	4	1	95	3	1	2E+05
	3	0.619	0.053	0.019	101	3	1	95	3	1	
	6	0.479	0.055	0.022	75	3	1	57	3	1	
	9	0.297	0.053	0.019	52	3	1	10	3	1	
	12	0.103	0.044	**0.016**	17	2	1	3	2	0	
40	0	2.877	0.791	1.094	775	168	158	240	40	36	8E+07
	3	2.426	0.663	0.894	330	57	52	240	40	36	
	6	1.409	0.445	0.544	183	35	32	107	28	24	
	9	0.739	0.301	0.338	119	24	21	20	12	10	
	12	0.189	**0.142**	0.149	28	9	7	3	4	3	
60	0	6.827	4.717	7.427	1,975	1,159	1,148	362	163	159	6E+09
	3	5.560	3.908	6.018	673	351	346	362	163	159	
	6	2.809	2.219	3.149	347	184	180	151	89	86	
	9	1.334	1.196	1.535	204	106	102	19	25	23	
	12	**0.255**	**0.331**	0.425	36	23	22	3	5	5	
80	0	14.181	14.199	21.791	4,283	3,704	3,703	370	278	277	1E+11
	3	11.334	11.797	17.916	1,320	1,109	1,107	370	278	277	
	6	5.305	6.286	9.061	626	519	518	128	98	97	
	9	2.204	2.890	3.725	336	274	273	17	21	20	
	12	**0.318**	0.543	0.714	44	40	40	1	3	3	
100	0	23.595	27.129	41.744	7,451	7,451	7,451	0	0	0	1E+12
	3	19.050	22.842	34.800	2,161	2,161	2,161	0	0	0	
	6	8.325	11.528	16.636	957	957	957	0	0	0	
	9	3.153	4.863	6.255	484	484	484	0	0	0	
	12	**0.366**	0.681	0.884	51	51	51	0	0	0	

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.

Table 5

AND/OR search algorithms (2): random networks; induced width 28 and 41; pseudo tree depth 38 and 51; averages over 20 instances

t	i	Time		Nodes (*1000)		Dead-ends (*1000)		#sol
		AO-FC	AO-RFC	AO-FC	AO-RFC	AO-FC	AO-RFC	
N=100, K=2, R=10, P=2, C=30, S=3, 20 instances, w*=28, h=38								
10	0	**1.743**	**1.743**	15	15	15	15	0
	10	1.748	1.746	15	15	15	15	
	20	1.773	1.784	15	15	15	15	
20	0	**3.193**	3.201	28	28	28	28	0
	10	3.195	3.200	28	28	28	28	
	20	3.276	3.273	28	28	28	28	
30	0	69.585	62.911	805	659	805	659	0
	10	69.803	**62.908**	805	659	805	659	
	20	69.275	63.055	805	659	687	659	
N=100, K=2, R=5, P=3, C=40, S=3, 20 instances, w*=41, h=51								
10	0	1.251	0.382	7	2	7	2	0
	10	1.249	**0.379**	7	2	7	2	
	20	1.265	0.386	7	2	7	2	
20	0	22.992	**15.955**	164	113	163	111	0
	10	22.994	15.978	162	110	162	111	
	20	22.999	16.047	162	110	162	110	
30	0	253.289	43.255	2093	351	2046	304	0
	10	254.250	**42.858**	2026	283	2032	289	
	20	253.439	43.228	2020	278	2026	283	

Table 6

AND/OR search vs. bucket elimination; random networks; averages over 20 instances

t	i	Time			Nodes (*1000)		Dead-ends (*1000)		#sol
		BE	AO-FC	AO-RFC	AO-FC	AO-RFC	AO-FC	AO-RFC	
N=70, K=2, R=5, P=2, C=30, S=3, 20 instances, w*=22, h=30									
40	0	26.4	2.0	1.3	49	21	35	19	0
	10		1.9	**1.2**	30	18	29	18	
	20		1.9	1.3	26	17	21	16	
50	0		30.7	35.6	2,883	2,708	1,096	1,032	1E+12
	10		18.6	18.9	557	512	342	302	
	20		12.4	**12.1**	245	216	146	130	
60	0		396.8	511.4	51,223	50,089	13,200	12,845	7E+14
	10		167.9	182.5	5,881	5,708	2,319	2,241	
	20		**80.5**	83.6	1,723	1,655	718	697	
N=60, K=2, R=5, P=2, C=40, S=3, 20 instances, w*=23, h=31									
40	0	67.3	0.7	**0.6**	9	9	8	7	0
	10		0.6	0.6	6	5	5	5	
	20		0.6	0.6	5	5	4	4	
50	0		3.2	3.0	58	55	41	38	6E+04
	10		3.0	2.8	31	28	28	25	
	20		2.7	**2.6**	25	23	20	18	
60	0		65.2	70.2	2,302	2,292	1,206	1,195	8E+08
	10		54.1	56.4	791	781	660	649	
	20		**39.6**	40.7	459	449	319	309	

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.

Table 7

AND/OR search vs. OR search vs. bucket elimination; random networks; averages over 20 instances

N=20, K=3, C=20, S=4, 20 instances, w*=9, h=14															
t		**10%**		**20%**		**30%**		**40%**		**50%**		**60%**		**70%**	
#solutions		**0**		**0**		**0**		**49**		**3,842**		**126,957**		**2,856,064**	
Time (seconds)															
	BE	0.10110		0.10155		0.10115		0.10025		**0.10000**		**0.08970**		**0.08805**	
i=0	A/O FC	0.00650		0.01250		**0.02450**		**0.06555**		0.22940		1.09355		5.81740	
	A/O RFC	**0.00350**		**0.01005**		0.02555		0.07660		0.27490		1.33295		6.94850	
	OR FC	0.00505		0.01200		0.02755		0.08670		0.52620		5.49720		65.68775	
	OR RFC	0.00400		0.01255		0.02800		0.09870		0.56040		5.72635		67.94275	
i=3	A/O FC	0.00550		0.01210		0.02555		**0.06410**		0.22925		1.09505		5.79485	
	A/O RFC	**0.00300**		0.01305		0.02550		0.07810		0.27850		1.33705		6.90190	
	OR FC	0.00555		0.01250		0.02750		0.08765		0.52405		5.48500		65.83190	
	OR RFC	0.00400		**0.01000**		0.02810		0.09820		0.56400		5.72880		67.98520	
i=6	A/O FC	0.00500		0.01250		**0.02405**		**0.06455**		0.21370		0.91375		4.33875	
	A/O RFC	0.00500		0.01100		0.02750		0.07555		0.25930		1.09625		5.08375	
	OR FC	0.00450		0.01250		0.02960		0.08860		0.49920		4.66985		49.77530	
	OR RFC	**0.00300**		0.01050		0.03200		0.09805		0.53625		4.87520		51.24910	
i=9	A/O FC	0.00455		0.01155		0.02500		**0.06405**		0.17240		0.48865		1.22135	
	A/O RFC	0.00450		**0.00950**		0.02600		0.07310		0.20530		0.58830		1.46265	
	OR FC	0.00550		0.01355		0.02950		0.08160		0.40010		2.98980		23.39555	
	OR RFC	0.00450		0.01150		0.03020		0.09415		0.43620		3.15515		24.25300	
Number of expanded nodes (# n) / Number of dead-ends (# d)															
		#n	#d	#n	#d	#n	#d	#n	#d	#n	#d	#n	#d	#n	#d
i=0	A/O FC	225	453	518	1032	1192	2330	3552	6579	16003	24402	106651	119059	735153	553820
	A/O RFC	154	311	387	771	1052	2056	3407	6307	15737	23987	106617	118989	735153	553820
	OR FC	225	453	519	1040	1203	2408	3810	7476	28079	44634	414463	448055	6533674	4499159
	OR RFC	154	311	387	777	1062	2126	3664	7183	27801	44078	414428	447986	6533674	4499159
i=3	A/O FC	225	453	518	1032	1192	2330	3552	6579	16003	24402	106651	119059	735153	553820

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.

N=20, K=3, C=20, S=4, 20 instances, w*=9, h=14															
t		**10%**		**20%**		**30%**		**40%**		**50%**		**60%**		**70%**	
#solutions		**0**		**0**		**0**		**49**		**3,842**		**126,957**		**2,856,064**	
	A/O RFC	154	311	387	771	1052	2056	3407	6307	15737	23987	106617	118989	735153	553820
	OR FC	225	453	519	1040	1203	2408	3810	7476	28079	44634	414463	448055	6533674	4499159
	OR RFC	154	311	387	777	1062	2126	3664	7183	27801	44078	414428	447986	6533674	4499159
i=6	A/O FC	224	451	512	1021	1162	2285	3306	6269	12765	21129	70273	88589	436554	368111
	A/O RFC	154	311	384	765	1028	2019	3175	6012	12562	20776	70238	88519	436554	368111
	OR FC	225	453	519	1040	1203	2408	3764	7418	24700	41194	294525	349350	3931078	3068920
	OR RFC	154	311	387	777	1062	2126	3618	7124	24422	40638	294491	349281	3931078	3068920
i=9	A/O FC	224	449	499	978	1093	2112	2883	5288	8873	14193	28038	33210	**79946**	**60144**
	A/O RFC	153	308	371	722	962	1857	2761	5063	8705	13899	28003	33141	79946	60144
	OR FC	225	453	518	1032	1192	2333	3604	6874	18729	30992	166912	203854	**1516976**	**1259120**
	OR RFC	154	311	387	771	1052	2058	3461	6597	18457	30477	166877	203784	1516976	1259120

Table 8

AND/OR search vs. OR search vs. bucket elimination; random networks; averages over 20 instances

N=40, K=3, C=50, S=3, 20 instances, w*=13, h=20													
t		**10%**		**20%**		**30%**		**40%**		**50%**		**60%**	
# solutions		**0**		**0**		**0**		**0**		**46582**		**147898575**	
Time (seconds)													
	BE	8.674		8.714		8.889		8.709		8.531		**8.637**	
i=0	A/O FC	**0.011**		**0.030**		**0.110**		**0.454**		**3.129**		32.931	
	OR FC	**0.009**		0.031		0.113		0.511		14.615		9737.823	
i=3	A/O FC	0.011		0.031		0.111		**0.453**		**3.103**		31.277	
	OR FC	0.009		0.030		0.112		0.509		14.474		9027.365	
i=6	A/O FC	0.011		0.029		0.110		0.454		**3.006**		25.140	
	OR FC	0.010		0.032		0.113		0.508		13.842		7293.472	
i=9	A/O FC	0.010		0.030		0.114		0.453		**2.895**		21.558	
	OR FC	0.010		0.031		0.111		0.509		12.336		5809.917	
i=13	A/O FC	0.011		0.030		0.111		0.457		**2.605**		**11.974**	
	OR FC	0.010		0.032		0.123		0.494		8.703		1170.203	
Number of expanded nodes (# n) / Number of dead-ends (# d)													
		# n	# d	# n	# d	# n	# d	# n	# d	# n	# d	# n	# d
i=0	A/OFC	78	159	265	533	999	1994	4735	9229	60163	101135	1601674	1711947
	ORFC	78	159	265	533	1000	2003	4947	9897	273547	407350	384120807	324545908
i=3	A/OFC	78	159	265	533	986	1990	4525	9166	46763	98413	689154	1625075
	ORFC	78	159	265	533	1000	2003	4947	9897	224739	399210	228667363	287701079
i=6	A/OFC	78	159	265	533	981	1971	4467	8991	41876	85583	487320	917612
	ORFC	78	159	265	533	1000	2003	4947	9897	185422	329754	141610990	208159068
i=9	A/OFC	78	159	265	533	981	1958	4451	8866	37314	70337	362024	580325
	ORFC	78	159	265	533	1000	2003	4947	9897	147329	270446	102316417	135655353
i=13	A/OFC	78	159	265	533	981	1955	4415	8533	30610	50228	**170827**	**181157**
	ORFC	78	159	265	533	999	1994	4761	9283	99923	176630	**16210028**	**20018823**

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.

Table 9

The impact of caching (A/O FC); random networks; averages over 20 instances

N=40, K=2, C=40, S=3, 20 instances, w*=10, h=17											
t		**10%**	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
#sol		**0**	**0**	**0**	**0**	**0**	**13,533**	**2,414,724**	**190,430,000**	**21,549,650,000**	**1,099,511,627,776**
Time											
A/O FC	i=0	0.000	0.001	0.002	0.005	0.011	0.065	0.289	1.931	7.979	30.094
	i=3	0.001	0.002	0.002	0.003	0.008	0.060	0.253	1.525	6.062	22.340
	i=6	0.001	0.001	0.004	0.003	0.009	0.052	0.182	0.883	2.873	8.847
	i=10	**0.000**	**0.001**	**0.003**	**0.004**	**0.010**	**0.038**	**0.110**	**0.343**	**0.587**	**0.985**
Number of nodes											
A/O FC	i=0	11	17	32	55	166	3078	22273	204562	988136	4145934
	i=3	11	17	32	55	155	1503	8747	57778	236466	870866
	i=6	11	17	32	55	148	975	4292	24542	95394	298236
	i=10	**11**	**17**	**32**	**55**	**135**	**746**	**2365**	**8646**	**15050**	**25717**
Number of dead-ends											
A/O FC	i=0	13	19	34	57	162	1978	10298	57678	134324	0
	i=3	13	19	34	57	159	1662	8569	45336	92263	0
	i=6	13	19	34	56	149	974	3721	13655	19257	0
	i=10	**13**	**19**	**34**	**55**	**125**	**533**	**1312**	**2313**	**1887**	**0**

Ann Math Artif Intell. Author manuscript; available in PMC 2010 October 25.