First-Order Bayes-Ball

Wannes Meert, Nima Taghipour, and Hendrik Blockeel

Katholieke Universiteit Leuven, Department of Computer Science,
Celestijnenlaan 200A, 3001 Heverlee, Belgium

Abstract. Efficient probabilistic inference is key to the success of statistical relational learning. One issue that increases the cost of inference is the presence of irrelevant random variables. The Bayes-ball algorithm can identify the requisite variables in a propositional Bayesian network and thus ignore irrelevant variables. This paper presents a lifted version of Bayes-ball, which works directly on the first-order level, and shows how this algorithm applies to (lifted) inference in directed first-order probabilistic models.

1 Introduction

Probabilistic logic models [2] bring the expressive power of first-order logic to probabilistic models, enabling them to capture both the relational structure and the uncertainty present in data. Efficient inference, however, is a bottleneck in these models, affecting also the cost of learning these models from data. Many attempts have been made to make inference more efficient in these formalisms, including lifted inference methods which try to exploit the symmetries present in the first-order probabilistic model [9,10,8,12]. In this paper, we make use of the first-order structure to focus on the issue of irrelevant random variables in probabilistic logic inference. To answer a specific probabilistic query, there is a minimum set of random variables which are required to be included in the computations. This set is called the *minimal requisite network* (MRN) [11]. Inference becomes more efficient by restricting computations to the MRN.

Bayes-ball [11] is an efficient algorithm that finds the MRN for inference in (propositional) Bayesian networks. The naive way of applying Bayes-ball to a probabilistic logic model is to ground the entire model and apply Bayes-ball on the resulting propositional network. This can be computationally expensive because the grounded network is often large or infinite, and its construction can be a significant part of the total inference cost.

Another way to compute the MRN for a probabilistic logic model consists of two steps: In the first step, all logical proofs for the query and for all evidence atoms are computed (e.g., using SLD resolution [2, Ch.10]), and a network is built using all ground clauses that are used therein. In the second step, the MRN is computed by applying Bayes-ball to this network. The second step is necessary since some atoms encountered in certain proofs of an evidence atom may be D-separated [11] from the query. This method has the disadvantage that it initially computes a Bayesian network that may be larger than the MRN.

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part II, LNAI 6322, pp. 369–384, 2010.

None of the mentioned methods take full advantage of the first-order representation of the probabilistic logic model. First-order probabilistic models introduce many random variables which are essentially identical with respect to inference, and hence also share the same status of relevance for a specific probabilistic query. We propose a first-order version of the Bayes-ball algorithm called first-order Bayes-ball (FOBB) that exploits these symmetries to efficiently compute the MRN for probabilistic logic inference. FOBB works directly at the first-order level while building the MRN; there is no need to build the ground network in the beginning. This algorithm treats indistinguishable random variables as one first-order atom and can process them in one single step (performing only one instance of identical operations).

Another contribution of FOBB is in the *first-order representation of the MRN*. This is valuable for lifted inference algorithms, which are designed specifically to make use of the first-order representation. The existing methods for computing the MRN all produce a ground MRN which does not preserve the first-order structure of the original model. FOBB, on the other hand, produces a first-order MRN for each query. This network can be used by lifted inference algorithms and can also make lifted inference more efficient by removing unnecessary but costly operations on the irrelevant parts of the model (such as shattering [8,4]). To the best of our knowledge the issue of relevance has not been addressed in lifted inference settings before.

While FOBB is a general method that applies to several directed probabilistic logic formalisms such as BLPs [3] and CP-logic [6], we illustrate it for the particular case of parametrized Bayesian networks [9], because of the intuitive relationship of this formalism to Bayesian networks.

In section 2 we review the Bayes-ball algorithm and parametrized Bayesian networks. Section 3 introduces the FOBB algorithm. Experiments are presented in Section 4. Section 5 contains the conclusions.

2 Preliminaries

2.1 Bayes-Ball

Bayes-ball [11] identifies the MRN of a Bayesian network for a given set of query and evidence nodes. It is based on the analogy of bouncing balls that travel over the edges of the Bayesian network (Fig. 1). In this analogy a visit from node a to b is compared to passing a ball from a to b. The balls start at the query nodes (each query node receives a ball). Upon reaching each node, a ball may *pass through*, *bounce back* and/or be *blocked*. The action chosen depends on the direction from which it came and on whether the node is probabilistically or deterministically dependent on its parents (based on D-separation). When a ball is to be passed to multiple nodes, it means that a visit to each of those nodes is put in a schedule. At each step a visit is selected from the schedule and processed.

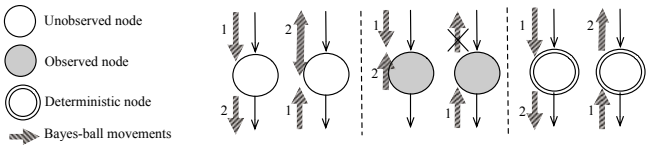

Fig. 1. Different actions performed on the Bayes-ball depending on the type of the node and direction from which the ball comes

The rules by which the balls moves through the network can be summarized as follows:

(a) An unobserved probabilistic node passes balls from parents on, that is, if such a node receives the ball from a parent it passes the ball to all its children. When such a node receives the ball from a child, it passes the ball to both its parents and children.
(b) An observed node bounces balls back from parents, that is, upon receiving the ball from a parent, such a node passes the ball to all its parents. However, an observed node blocks balls from children, that is, the ball is not passed on anymore from this node.
(c) A functional unobserved node always passes balls through, that is, it passes the ball coming from parents to children and vice versa.

Nodes are marked at each visit of a ball, depending on the type of action performed on the ball: when the ball is passed from a node to its parents (children), the node receives a mark on top (bottom). These marks help the algorithm avoid repeating the same action, and guarantee the termination of the algorithm. Having a mark on top (bottom) indicates that there is no need to visit the parents (children) anymore. In the end, these marks indicate the relevance of each node: the MRN consists of all the nodes marked on the top together with the set of evidence atoms visited during the algorithm.

2.2 Parametrized Bayesian Networks

To illustrate FOBB we represent probabilistic logic models as *parametrized Bayesian networks* (PBNs) [9]. In such a model, random variables of a Bayesian network are represented by (ground) atoms, and definite clauses are used to capture the structure of the network. In this way, a first-order atom represents a class of random variables, and a first-order clause represents repeating structures in the Bayesian network. An example PBN is presented in Fig. 2.a.

Each predicate p takes on values from a specific range $Range(p)$, which contains the possible states of the random variables represented by this predicate. Moreover, each argument of a predicate takes values of a specific type. A PBN includes *domain declarations* defining the set of constants of each type, and *functor declarations* assigning a type to each argument of each predicate.

A PBN also consists of a *theory*, which captures the structure of the probabilistic model through a set of *Bayesian clauses*. A Bayesian clause bc is an expression of the form: [1]

$$(bc)\ \forall X_1, \ldots, X_n : C;\ h|b_1, \ldots, b_n$$

with $h, b_1, \ldots, b_n$ first-order atoms with logic variables $X_1, \ldots, X_n$ as arguments. Each logic variable X_i is implicitly assigned to a domain, considering the *functor declarations*. The constraint C restricts all possible groundings of $X_1, \ldots, X_n$ to a subset of the Cartesian product of their respective domains. We define $head(bc) = h$ and $body(bc) = \bigcup_i b_i$. A Bayesian clause specifies that for each substitution $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ that grounds the clause and is in accordance with C, the random variable $h\theta$ depends on $b_1\theta, \ldots, b_n\theta$.

Each Bayesian clause is associated with a conditional probability distribution (CPD) $P(h|b_1, \ldots, b_n)$, which specifies the same distribution $P(h\theta|b_1\theta, \ldots, b_n\theta)$ for every ground instance of the clause. If a ground atom appears in the head of more than one ground clause, then a *combining rule* is used to obtain the desired probability distribution from the CPDs associated to those clauses. Each predicate has an associated combining rule.

2.3 Equivalent Bayesian Network

Like most directed probabilistic first-order models, a PBN can be transformed to a Bayesian network. Here, we use a transformation to what we call an equivalent Bayesian network (EBN), similar to the EBNs in Meert et al [6]. The EBN is a regular Bayesian network, but we consider it as a bipartite graph containing two types of nodes: *atom nodes* and *factor nodes* (see Fig. 2.b).

Atom nodes correspond to the random variables; there is an atom node in the EBN for each atom in the grounding of the theory.

Factor nodes explicitly capture the factorization declared by the Bayesian clauses. For each clause bc and each grounding substitution θ complying with the constraint C of bc, there is a factor node corresponding to $bc\theta$ in the EBN. We denote factor nodes by atoms too. Each Bayesian clause bc is associated to an n-ary predicate bc', with $n = |Var(bc)|$. This predicate has the same domain as $head(bc)$. The factor node corresponding to ground clause $bc\theta$ can be represented by the atom $bc'\theta$.

There are edges between factor nodes and atom nodes, but no edges between nodes of the same type. The edges of the network can be fully described with this rule: Each factor node $bc'\theta$, corresponding to a ground clause $bc\theta$, has as parents all the atoms in $body(bc\theta)$, and is the parent of the atom $head(bc\theta)$.

The nodes and edges represent the structure of the EBN. In the quantitative component of the EBN, the CPD of each clause bc is associated to the corresponding

[1] We use a slightly different version of Poole's PBN [9]: here each rule is associated with an entire CPD, instead of declaring the probability of one specific combination of values for a node and its parents. This idea is introduced in BLPs [3], from which we also borrow the term *Bayesian clause*.

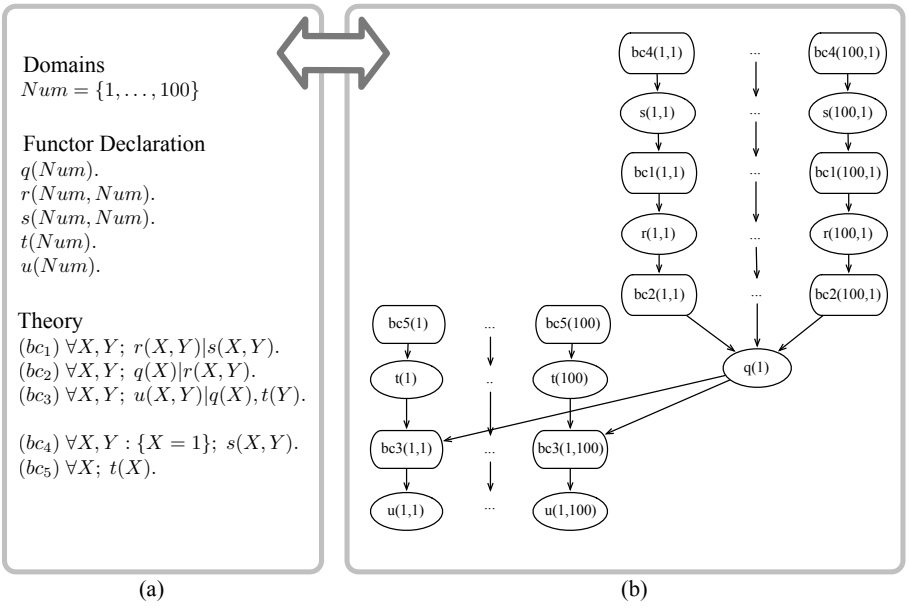

Fig. 2. (a) Parametrized Bayesian network (b) Equivalent Bayesian network having the same probability distribution as the theory. Bayes-ball can be used on such a Bayesian network to find the MRN given a query and evidence.

factor node bc'. This assignment of CPD to factor nodes is straightforward since by definition factor nodes have the same domain as $head(bc)$. The CPD on the atom nodes is a deterministic function of the parent factor nodes, implementing the effect of the combining rule associated to the atom. In this work, we consider combining rules that represent *independent causation*, such as *noisy-or*, *-and*, *-max*, and *-min*.

3 First-Order Bayes-Ball

FOBB is based on the same principles as Bayes-ball, building upon the transformability of a probabilistic logic model to an EBN. Its main advantage is the possibility to perform some steps at the first-order level. That is, several nodes can be represented by what we call a *first-order node*, and be visited in one single step. After a definition of first-order nodes and related operations on such nodes, we show the main features of the algorithm through an example; this is followed by a more detailed description.

Definition 1 ***(Constraint)*** *Having logic variables* $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$*, with* $D(X_i)$ *the associated domain of* X_i*, a constraint* C *on* $\mathbf{X}$ *is a relation on* $\mathbf{X}$*, indicating a subset of the Cartesian product* $D(\mathbf{X}) = \times_i D(X_i)$*.*

There is no restriction on how to represent and store the constraints. The choice of representation, however, affects the efficiency of the algorithm. For example,

storing them as ground tuples would cancel the advantages of FOBB over Bayes-ball. For the implementation we opted to store constraints as decision trees with set membership tests in the nodes. This representation is different from that used in [9] for PBNs and other work about lifted inference, where a constraint is a set of (in)equalities involving logic variables and constants. One such conjunction is equivalent to one branch in our decision tree.

Definition 2 ***(First-order node)*** *A first order node $\mathcal{F}$ is a pair (p, C), where $p = a(X_1, \ldots, X_n)$ is a first-order atom, and C is a constraint on logic variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. Each first-order node $\mathcal{F} = (p, C)$ represents the set of ground random variables $p\theta$, where $\mathbf{X}\theta \in C$. We denote the set of (ground) random variables represented by $\mathcal{F}$ as $RV(\mathcal{F})$.*

For two first-order nodes $\mathcal{F}_1 = (p, C_1)$ and $\mathcal{F}_2 = (p, C_2)$, we define:

1. $\mathcal{F}_1 \subseteq \mathcal{F}_2$ iff $RV(\mathcal{F}_1) \subseteq RV(\mathcal{F}_2)$ iff $C_1 \subseteq C_2$.
2. $\mathcal{F}_1 \Delta \mathcal{F}_2 = \mathcal{F}'$ iff $\mathcal{F}' = (p, C_1 \Delta C_2)$, for $\Delta \in \{\cap, \cup, \setminus\}$.

Definition 3 ***(Splitting)*** *The result of splitting a first-order node $\mathcal{F} = (p, C)$ is a set of first-order nodes $\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, where each $\mathcal{F}_i = (p, C_i)$, and such that $\bigcup_i C_i = C$ and $\bigcap_i C_i = \emptyset$.*

FOBB also uses the operation of *projection* on constraints:

Definition 4 ***(Projection)*** *Let C be a constraint on logic variables $\mathbf{X}$. Projection of C on a subset of its variables $\mathbf{Y} \subseteq \mathbf{X}$ is given by the constraint $\pi_Y(C) = \{\mathbf{y} = (y_1, \ldots, y_{|Y|}) | \exists \overline{\mathbf{y}} \in C$, and $\overline{\mathbf{y}}$ is an extension of $\mathbf{y}\}$.*

3.1 Overview

FOBB computes the MRN for a probabilistic query $P(Q|E)$ on the probabilistic logic model M. It is assumed that the query atoms Q are ground and that the theory T of M has a finite grounding. The outer structure of Alg. 1 closely resembles the original Bayes-ball (see [11]), the main differences are that it works with first-order nodes instead of ground nodes, and that it uses the given first-order probabilistic logic model to compute the parents and children of a first-order node.

We use an example to illustrate FOBB and compare it with the original Bayes-ball: Suppose we need to compute $P(Q|E)$ where $Q = \{q(1)\}$ and $E = \{r(1,1), r(1,2), u(1,1), \ldots, u(1,10)\}$, given the theory in Fig. 2.

Similar to Bayes-ball, FOBB schedules the nodes which are to be visited. Instead of scheduling ground nodes to visit, FOBB schedules first-order nodes. Each entry in the schedule is represented by a tuple $\langle \mathcal{F}, direction \rangle$, containing a first-order node $\mathcal{F}$, and the *direction* of the visit ($fromChild$ or $fromParent$). This entry stands for a visit to each node in $RV(\mathcal{F})$, in Bayes-ball. In the beginning, FOBB starts by scheduling the query: $\langle (q(X), \{X{=}1\}), fromChild \rangle$. Next, FOBB retrieves this tuple from the schedule and computes its parents by matching $(q(X), \{X{=}1\})$ to the heads of clauses of T. In this case, only Bayesian clause

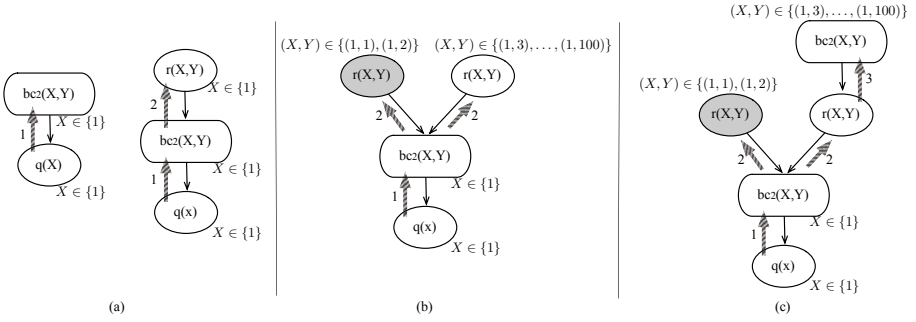

Fig. 3. Illustration of the FOBB algorithm as explained in Sec. 3.1. (a) A ball is passed on from a node to its ancestors. (b) When not all ground nodes represented by a first-order node respond identically to the ball, the node is split up. In this case, part of the nodes represented by the first-order node are observed and do not pass on the ball. (c) The first-order node that represents the ground nodes that are not observed and do not yet have top mark passes on the ball to its ancestors.

bc_2 matches, so FOBB schedules the factor node $\langle(bc_2'(X,Y),\{X{=}1\}),\mathit{fromChild}\rangle$. The constraint $\{X{=}1\}$ makes sure that only the subset of $bc_2'(X,Y)$ that are the parents of the query, are included. We will elaborate on this later. Note that the scheduled first-order node actually represents multiple nodes in the EBN (due to the free variable Y). This is shown in the first step in Fig. 3.

When a subset $\mathcal{F}'$ of the nodes in $\mathcal{F}$ interact differently with the rest of the network (e.g. they are observed variables), we need to separate them from other nodes in $\mathcal{F}$. We call this operation *splitting*, following Poole [9]. Continuing our example, when $\mathcal{F} = (r(X,Y),\{X{=}1\})$ receives the ball (from child $(bc_2'(X,Y),\{X{=}1\})$) it contains nodes $r(1,1)$,$r(1,2)$ which are evidence while the rest of the nodes are not observed. Hence, the algorithm splits the original first-order node $\mathcal{F}$ into $\mathcal{F}_{\neg E} = (r(X,Y),\{X = 1, Y \notin \{1,2\}\})$, consisting of unobserved nodes, and $\mathcal{F}_E = (r(X,Y),\{X = 1, Y \in \{1,2\}\})$, consisting of the evidence. Now, when $\mathcal{F}_{\neg E}$ receives the ball it passes the ball to its parents and children, while $\mathcal{F}_E$, which contains only observed nodes, blocks the ball. The splitting operation will be defined in detail later.

Passing the ball to the children involves similar operations as sending the ball to parents. For example, to find children of $(q(x),\{x = 1\})$, we need to find clauses which have this atom in their body. We see it only appears in the body of clause bc_3, and so its children would be $(bc_3'(X,Y),\{X = 1\})$.

Bayes-ball assigns *top* and/or *bottom* marks to the nodes it visits. FOBB also keeps track of marks, but here these are marks for first-order nodes. It stores the marks as pairs $\langle\mathcal{F},\mathbf{M}\rangle$ in a mark table, with $\mathcal{F}$ being a first-order node, and $\mathbf{M}$ a set of marks. For example, the initial marks for the query atom and for $bc_1'(X,Y)$ are stored as $\langle(q(X),\{X = 1\}),\{top\}\rangle$ and $\langle(bc_1'(X,Y),\{X{=}1\}),\{top\}\rangle$, showing these two first-order nodes have passed the ball to their respective parents.

During the execution of the algorithm, it is possible that we have to split an atom in the marks table. This happens when a subset of the nodes

presented by a first-order node need to be assigned additional marks. In our example, $\mathcal{F}_1 = (bc'_3(X,Y), \{X = 1\})$ passes the ball to its children resulting in $\langle \mathcal{F}_1, \{bottom\} \rangle$ being registered in the marks table. Later when the ball is passed up from $(u(X,Y), \{X = 1, Y \in \{1,\ldots,10\}\})$ to $\mathcal{F}'_1 = (bc'_3(X,Y), \{X = 1, Y \in \{1,\ldots,10\}\})$ this node should in turn pass the ball up and receive a mark on top. For this reason we need to first split $\mathcal{F}_1$, keeping $\langle \mathcal{F}_1 \setminus \mathcal{F}'_1, \{bottom\} \rangle$ from the original first-order node and adding $\langle \mathcal{F}'_1, \{top, bottom\} \rangle$ to the marks table, for the subset which passes the ball up.

When FOBB terminates, all the first-order *factor* nodes marked at the top, together with the visited evidence atoms constitute the MRN.

3.2 The Algorithm

The FOBB algorithm uses the same set of rules as Bayes-ball to send balls through the network, visiting the possibly relevant nodes. The main difference is that the balls are not passed between nodes as we know them from Bayesian networks but between first-order nodes that aggregate multiple ground nodes in one higher level node. This way FOBB can perform multiple identical operations on $RV(\mathcal{F})$ in one single step instead of performing $|RV(\mathcal{F})|$ equivalent steps in Bayes-ball.

FOBB schedules visits to a group of ground nodes aggregated in a first-order node, searches for parents and children of such a first-order node and assigns marks to first-order nodes. The aim is to keep the nodes as aggregated as possible, but when a subset of the nodes behave differently it is necessary to split the first-order node and treat those subsets separately. The splitting happens when needed during the execution of the algorithm. Next, we illustrate how the operations in FOBB differ from those in Bayes-ball:

Initialization
In the initialization of the algorithm, all the query atoms are added to the schedule as if they were visited from child. For this we need to represent the query nodes as first-order nodes. This is done by the GetFONode method that takes as input a ground atom $q(a_1,\ldots,a_n)$ and outputs a first-order node (p, C) with $p = q(X_1,\ldots,X_n)$ and $C = \{X_i = a_i\}$.

Scheduling visits
Where Bayes-ball has a schedule with pairs of nodes and directions to keep track of scheduled visits, FOBB utilizes a schedule containing pairs of first-order nodes and directions. An entry in the schedule containing first-order node $\mathcal{F}$ stands for a set of visits to the ground nodes in $RV(\mathcal{F})$. When a ground node in Bayes-ball receives a ball it will respond according to the rules in Sec. 2.1. In FOBB, however, it is possible that not all ground nodes represented by $\mathcal{F}$ pass the ball in the same way. This happens when some of the nodes are part of the evidence, or when not all the nodes have the same marks. In this case the first-order node $\mathcal{F}$ is split into new first-order nodes representing subsets of the ground nodes in $\mathcal{F}$ that pass the ball identically.

Algorithm 1. FOBB(M, Q, E)

Input: M: probabilistic logic model, Q: set of ground query atoms, E: set of ground evidence atoms
Output: R: requisite network, E_R: requisite evidence

$S \leftarrow \emptyset$, $E_R \leftarrow \emptyset$
for each $q \in Q$ **do**
  $\mathcal{Q} = \text{GetFONode}(q)$; $S \leftarrow S \cup \langle \mathcal{Q}, \mathit{fromChild} \rangle$
while $S \neq \emptyset$ **do**
  pick and remove a visit $\langle \mathcal{F}, \mathit{direction} \rangle$ from S
  $(\mathcal{F}_E, \mathcal{F}_{\neg E}) \leftarrow \text{SplitOnEvidence}(\mathcal{F}, E)$
  if $\mathcal{F}_E \neq \emptyset$ **then**
    $E_R \leftarrow E_R \cup \mathcal{F}_E$
  if $\mathit{direction} = \mathit{fromChild} \wedge \mathcal{F}_{\neg E} \neq \emptyset$ **then** // Backward chaining
    $(\mathcal{F}_{\neg E}^{top}, \mathcal{F}_{\neg E}^{\neg top}) \leftarrow \text{SplitOnMark}(\mathcal{F}_{\neg E}, \mathit{top})$
    $\text{AddMark}(\mathcal{F}_{\neg E}^{\neg top}, \mathit{top})$
    for each $\mathcal{PA} \in \text{GetParents}(\mathcal{F}_{\neg E}^{\neg top}, T)$ **do**
      $S \leftarrow S \cup \langle \mathcal{PA}, \mathit{fromChild} \rangle$
    $(\mathcal{F}_{\neg E}^{btm}, \mathcal{F}_{\neg E}^{\neg btm}) \leftarrow \text{SplitOnMark}(\mathcal{F}_{\neg E}, \mathit{bottom})$
    if $\neg\text{Functional}(\mathcal{F}) \wedge \mathcal{F}_{\neg E}^{\neg btm} \neq \emptyset$ **then**
      $\text{AddMark}(\mathcal{F}_{\neg E}^{\neg btm}, \mathit{bottom})$
      for each $\mathcal{CH} \in \text{GetChildren}(\mathcal{F}_{\neg E}^{\neg btm}, T)$ **do**
        $S \leftarrow S \cup \langle \mathcal{CH}, \mathit{fromParent} \rangle$
  if $\mathit{direction} = \mathit{fromParent}$ **then** // Forward chaining
    if $\mathcal{F}_E \neq \emptyset$ **then**
      $E_R \leftarrow E_R \cup \mathcal{F}_E$
      $(\mathcal{F}_E^{top}, \mathcal{F}_E^{\neg top}) \leftarrow \text{SplitOnMark}(\mathcal{F}_E, \mathit{top})$
      $\text{AddMark}(\mathcal{F}_E^{\neg top}, \mathit{top})$
      for each $\mathcal{PA} \in \text{GetParents}(\mathcal{F}_E^{\neg top}, T)$ **do**
        $S \leftarrow S \cup \langle \mathcal{PA}, \mathit{fromChild} \rangle$
    if $\mathcal{F}_{\neg E} \neq \emptyset$ **then**
      $(\mathcal{F}_{\neg E}^{btm}, \mathcal{F}_{\neg E}^{\neg btm}) \leftarrow \text{SplitOnMark}(\mathcal{F}_{\neg E}, \mathit{bottom})$
      $\text{AddMark}(\mathcal{F}_{\neg E}^{\neg btm}, \mathit{bottom})$
      for each $\mathcal{CH} \in \text{GetChildren}(\mathcal{F}_{\neg E}^{\neg btm}, T)$ **do**
        $S \leftarrow S \cup \langle \mathcal{CH}, \mathit{fromParent} \rangle$
$R \leftarrow \{\mathcal{R} |\ \text{HasMark}(\mathcal{R}, \mathit{top})\}$
return (R, E_R)

For example, if $\mathcal{F}$ receives a ball from one of its children, those ground nodes in $RV(\mathcal{F})$ that are part of the evidence and those that already have a top mark do not need to pass the ball to their parents, while the other ones do.

In Alg. 1 two methods are used to split up a first-order node. First, SplitOnEvidence uses the *evidence* to split up first-order node $\mathcal{F}$ into $\mathcal{F}_E$ (containing all the evidence nodes in $\mathcal{F}$), and $\mathcal{F}_{\neg E}$ (containing non-evidence nodes of $\mathcal{F}$). All evidence atoms of predicate p can be represented as a first-order node (p, C_E), then $\mathcal{F}_E = (p, C \cap C_E)$ and $\mathcal{F}_{\neg E} = (p, C \setminus C_E)$. After $\mathcal{F}$ is split on evidence, first $\mathcal{F}_E$ receives the ball (and is added to the set of visited evidence atoms), and then $\mathcal{F}_{\neg E}$.

Second, the obtained first-order node $\mathcal{F}_{\neg E}$ is split further by SplitOnMark. To split $\mathcal{F} = (p, c)$ on mark m, we need to consult the marks table and find entries $\langle (p, C_i), \mathbf{M}_i \rangle$, such that $m \in \mathbf{M}_i$. Using the found entries, FOBB splits $\mathcal{F}$ into first-order node $\mathcal{F}^m = \bigcup_i \mathcal{F}_i = \bigcup_i (p, C \cap C_i)$, which has the mark m, and $\mathcal{F}^{\neg m} = \mathcal{F} \setminus \mathcal{F}^m$, which does not have the mark m.

After a first-order node is split into subsets which perform the same action to the ball, each subset can pass the ball to its parents/children.

Passing the Ball to Parents and Children.
Like in Bayes-ball, passing a ball from a node to its parents or children is done by following the outgoing or incoming links and scheduling a visit to the found nodes. However, since FOBB does not construct the fully grounded Bayesian network, the PBN must be used to find the parents and children of each first-order node. This is done differently for *atom* and *factor* nodes, considering the transformation of PBNs to their EBN.

The parents of each ground *atom* node $a \in RV(\mathcal{F})$ in the EBN are those factor nodes corresponding to the ground clauses which have a in their head. Thus, to find the parents of a first-order *atom* node $\mathcal{F} = (p, C)$, we first find the set of clauses $\{bc_1, \ldots, bc_k\}$, such that there is a (renaming) substitution θ_i where $head(bc_i) = p\theta_i$. Each bc_i can be represented by a first-order *factor* node $\mathcal{B}_i = (bc_i', C_i)$ where bc_i' is the atom associated to clause bc_i and each C_i is the constraint defined on the variables of bc_i in the PBN. Then, the parents of first-order node $\mathcal{F}$ in the EBN can be represented as first-order nodes $\mathcal{PA}_i = (bc_i', C_i')$, where each C_i' restricts $RV(\mathcal{PA}_i)$ to those which are parent of a node in $RV(\mathcal{F})$. Each constraint C_i' is equivalent to the relation acquired from the *natural join* $C_i \bowtie C$ of relations C_i and C (with the variables of C renamed according to substitution θ_i). This way, all the groundings of a clause bc_i that have an atom $a \in RV(F)$ in their head are captured by the first-order node $\mathcal{PA}_i$. Finding the children of an atom node is similar, only there the connected clauses are those which have a in their body.

When $\mathcal{F} = (bc', C)$ is a *factor* node, its parents are found by considering the body of the clause bc in the theory, to which bc' is associated. Let $body(bc) = \{b_1, \ldots, b_n\}$ and C_{bc} be the constraint associated to bc in the theory. Then, the parents of $\mathcal{F}$, are first-order nodes $\mathcal{B}_i = (b_i, C_i)$, where C_i restricts $RV(\mathcal{B}_i)$ to those which are parents of a node in $RV(\mathcal{F})$. Each $C_i = \pi_{X_i}(C \cap C_{bc})$ is the relation acquired from projecting $C \cap C_{bc}$ on variables $\mathbf{X_i} = Var(b_i)$. Similarly, for finding the children the head of the clause bc is considered instead of the body.

Having computed the parents $\mathcal{PA}_i$ (using GetParents), in the end an entry $\langle \mathcal{PA}_i, fromchild \rangle$ is registered in the schedule, for each $\mathcal{PA}_i \neq \emptyset$, to pass the ball to parents of $\mathcal{F}$. Similarly, to pass the ball to children an entry $\langle \mathcal{CH}_i, fromParent) \rangle$ is added to the schedule for the computed children $\mathcal{CH}_i$ (using GetChildren).

Assigning Marks
After passing the ball to the parents (children) of $\mathcal{F}$, FOBB needs to mark $\mathcal{F}$ on top (bottom). This can be naively done by adding $\langle \mathcal{F}, \{top\} \rangle$ to the marks

table. In this way, however, the marks table might include overlapping entries, that is, there might be a $\langle \mathcal{F}', \mathbf{M}' \rangle$ in the marks were $RV(\mathcal{F}) \cap RV(\mathcal{F}') \neq \emptyset$. In this case, $\mathbf{M}'$ contains only the *bottom* mark, since $\mathcal{F}$ is split on the top mark when retrieved from the schedule, guaranteeing that no subset of it has the top mark. Hence, the subset $\mathcal{F} \cap \mathcal{F}'$ should now have both the *top* and *bottom* marks, and should be grouped together. In general, when assigning a mark m to $\mathcal{F} = (p, C)$, if there is an overlapping mark $\mu = \langle \mathcal{F}' = (p, C'), \mathbf{M}' \rangle$ then we need to *split the marks*: First, μ is removed from the marks table, and then the marks $\mu_1 = \langle \mathcal{F} \cap \mathcal{F}', \mathbf{M}' \cup \{m\} \rangle$, $\mu_2 = \langle \mathcal{F} \setminus \mathcal{F}', \{m\} \rangle$, and $\mu_3 = \langle \mathcal{F}' \setminus \mathcal{F}, \mathbf{M}' \rangle$ are assigned instead. (Assigning these marks might result in further splits.) In this manner all the marks $\langle (p, C_i), \mathbf{M}_i \rangle$ form a partition on all the groundings of p which have been visited, such that all the nodes in each $\mathcal{F}_i = (p, C_i)$ have exactly the same marks.

3.3 Extension for Implicit Domains

The semantics of many probabilistic logic languages, such as BLPs [3] and CP-logic [6], declare an implicit domain for their logic variables. Although FOBB requires explicit domains, it can be extended to deduce the domains dynamically. Formally, we want to restrict the random variables to the *least Herbrand model* of the corresponding logic program. Intuitively, the set of ground nodes represented by a probabilistic logic model M are those which have a proof in M. To comply with these semantics, FOBB too needs to identify which random variables have a proof.

Most formalisms use some form of backward-chaining, such as SLD resolution, to find the least Herbrand model. The same idea can be adopted in FOBB. Note that Bayes-ball (and FOBB) effectively forms the backward-chains for each node from which the ball is passed to its parents and then its ancestors. Hence, FOBB is searching for proofs in a similar way to SLD resolution. The backward-chain ends whenever a root node or an evidence node receives the ball from a child. At this point we know whether this node has a proof. By chaining this information forward through the network, the nodes which have a proof can be identified.

In practice an extra mark, called a *proof mark*, is used to indicate what nodes have been proved. The MRN is then constituted by those nodes that have not only the top mark but also the proof mark. Also, the schedule has to give preference to those balls that have been passed on from proved parents.

4 Experiments

In our experiments, we investigated how the size of the domain of logic variables affects the search for the MRN, and how this MRN affects inference for probabilistic logic models. All experiments are performed on an Intel Pentium D CPU 2.80GHz processor with 1GB of memory available. FOBB itself is implemented in C++.

As a first experiment, we take the theory shown in Fig. 2 and compute the conditional probability of the atom $q(1)$ while varying the size of the domain

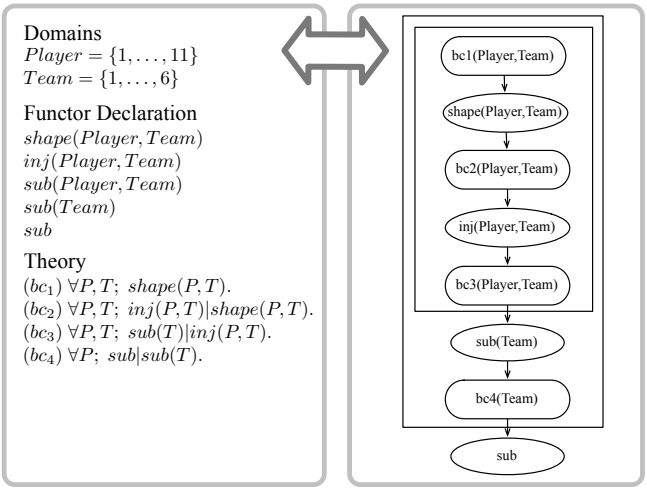

Fig. 4. (a) First-order model and (b) its equivalent belief network. The scopes of logic variables are indicated by the rectangles (plates).

Num. One third of the ground nodes is chosen at random and considered as observed. For the inference, while any Bayesian network inference could be applied, we used an implementation in C++ performing variable elimination with the optimization proposed in [1] to obtain linear inference for noisy-or nodes. Five approaches were used to obtain a Bayesian network from the original theory: (a) ground the entire network based on the domains; (b) ground the entire network and use Bayes-ball to limit the resulting equivalent Bayesian network to the MRN; (c) ground the network by means of SLD resolution (using Prolog and setting the query and evidence as goals); (d) ground by means of SLD resolution and use Bayes-ball to find the MRN; and (e) use FOBB to find the MRN directly from the theory and ground the MRN. For methods (a) and (b) the theory shown in Fig. 2 was transformed first to a ground theory and afterwards compiled to an equivalent Bayesian network. Methods (c) and (d) required to first ground the facts (clauses with empty body) according to the domains.

Fig. 5 shows the results of the first experiment. The bottom graph shows that FOBB is magnitudes faster in finding the (grounded) MRN than any of the other methods. The top graph shows that the complexity of performing inference grows faster than that of grounding. As a consequence, for large networks the approach used to find the MRN became of less importance in the total inference time. These results, however, confirm the importance of restricting the network to the MRN: The two methods that do not restrict the network, full grounding and SLD resolution, could not handle even the smaller networks and ran out of memory. This result motivates investigating the effect of restricting computations to the MRN in lifted inference, for which no method like Bayes-ball has been proposed to date. We applied FOBB to such a case in our second experiment.

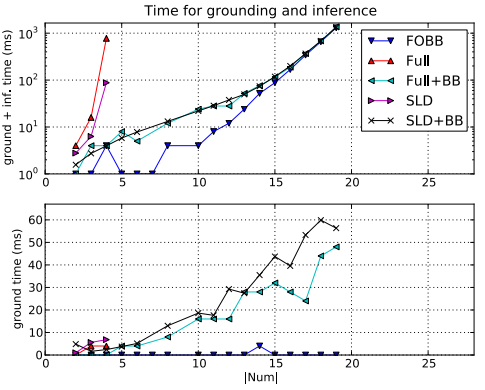

Fig. 5. Performance on the model in example in Fig. 2

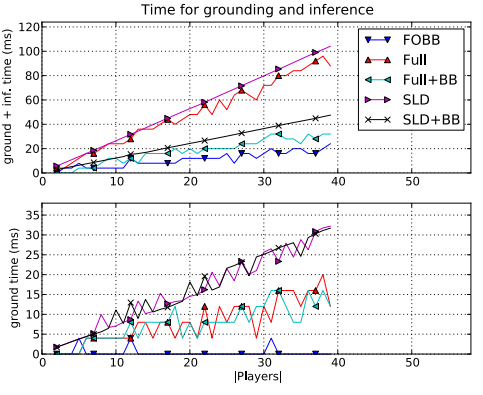

Fig. 6. Performance on the soccer example using propositional inference

In the second experiment we used the theory shown in Fig. 4. This theory is an extension of the theory used in [5] to benchmark lifted inference methods. This theory represents that whether a player in a soccer tournament is substituted during the tournament depends on whether he gets injured. The probability of an injury depends on the physical condition of the player. We compute the conditional probability of *substitution*(1) given that six teams participate, while varying the number of players in a team. For four of the teams there is evidence that some player has been injured.

For the results in Fig. 6 we used the same strategy as for the previous experiment. In addition to propositional inference we also used the lifted inference technique C-FOVE [8] available in BLOG [7] (Java) to calculate the conditional probability of the query. The factors are created based on the optimizations mentioned in [5].

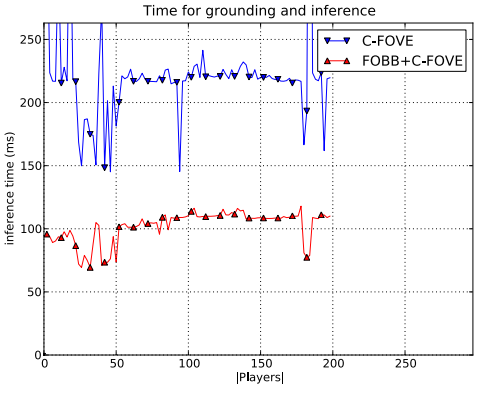

Fig. 7. Performance on the soccer example using lifted inference

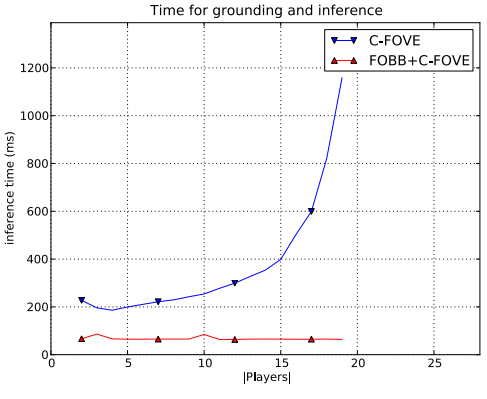

Fig. 8. Performance on the soccer example using lifted inference with a more complex interaction than noisy-or for one extra team

The soccer model is very symmetric and inference is therefore efficient. The results in Fig. 6 show that the complexity of grounding and inference are both linear. In this case the efficiency of grounding has a noticeable influence on the total inference time.

For this model, a lifted inference method can make abstraction of the domain size for performing probabilistic inference, and can therefore calculate the marginal probability of the query in constant time. This is shown in Fig. 7. FOBB allows us to find the MRN in a form that can be interpreted by a lifted inference method. With this combination, we can thus not only make abstraction of the domain size but also ignore non-requisite parts of the first-order probabilistic model. FOBB can have a greater influence on the inference when applied to to more comprehensive models, since it is possible to have non-requisite parts of arbitrary complexity. Such an effect can be observed, for example, when the model contains an extra team that uses a more complex combining rule for *sub* than noisy-or. This causes inference

to be exponential on the non-requisite parts. These unnecessary computations are avoided when using FOBB, as shown in Fig. 8.

5 Conclusions and Future Work

In this work, we presented a first-order version of Bayes-ball called FOBB, which finds the minimum relevant network for a given set of query and evidence atoms. The advantages of using FOBB are twofold; first, it is more efficient to find the ground network needed to calculate the probability of the query than current methods. Second, the resulting relevant network is first-order, permitting it to be used as input to lifted inference methods which have shown to offer magnitudes of gain in speed and memory. FOBB resembles the approach by Singla and Domingos [12] in aggregating ground nodes as one unit and building a *lifted network*; major differences are that FOBB is meant for directed graphs instead of undirected graphs, and that it is dependent on the query and not a compilation technique.

In general, empirical evaluations of lifted inference algorithms are done using simple first-order probabilistic models. FOBB is a valuable companion to existing lifted inference methods like the one proposed in [5] to handle more comprehensive and real-life models.

In the future, we want to investigate further the concept of the proof ball. This would make FOBB even more suited for probabilistic logic models that are based on logic programming.

Acknowledgements

Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) to Wannes Meert. GOA/08/008 'Probabilistic Logic Learning' to Nima Taghipour.

References

1. Díez, F.J., Galán, S.F.: Efficient computation for the noisy max. International Journal of Intelligent Systems 18(2), 165–177 (2003)
2. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning). The MIT Press, Cambridge (2007)
3. Kersting, K., De Raedt, L.: Bayesian Logic Programming: Theory and Tool. In: Getoor, L., Taskar, B. (eds.) An Introduction to Statistical Relational Learning, pp. 291–322. MIT Press, Cambridge (2007)
4. Kisynski, J., Poole, D.: Constraint processing in lifted probabilistic inference. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI (2009)
5. Kisynski, J., Poole, D.: Lifted aggregation in directed first-order probabilistic models. In: Proceedings of the 21th International Joint Conference on Artificial Intelligence, IJCAI (2009)

6. Meert, W., Struyf, J., Blockeel, H.: Learning ground CP-logic theories by leveraging Bayesian network learning techniques. Fundamenta Informaticae 89(1), 131–160 (2008)
7. Milch, B.: BLOG (2008), http://people.csail.mit.edu/milch/blog/
8. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P.: Lifted probabilistic inference with counting formulas. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI), pp. 1062–1608 (2008)
9. Poole, D.: First-order probabilistic inference. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI), pp. 985–991 (2003)
10. Braz, R.d.S., Amir, E., Roth, D.: Lifted first-order probabilistic inference. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1319–1325 (2005)
11. Shachter, R.D.: Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 480–487 (1998)
12. Singla, P., Domingos, P.: Lifted first-order belief propagation. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI), pp. 1094–1099 (2008)