A Monte-Carlo Algorithm for Probabilistic Propagation in Belief Networks based on Importance Sampling and Stratified Simulation Techniques

Luis D. Hernández
Dept. Informática y Sistemas
Universidad de Murcia
30071 - Espinardo - Murcia - SPAIN
e-mail: ldaniel@dif.um.es

Serafín Moral
Dept. Ciencias de la Computación e I.A.
Universidad de Granada
18071 - Granada - SPAIN
e-mail: smc@decsai.ugr.es

Antonio Salmerón
Dept. Estadística y Matemática Aplicada
Universidad de Almería
04120 - Almería - SPAIN
e-mail: asc@stat.ualm.es

July 10, 1997

Abstract

A class of Monte Carlo algorithms for probability propagation in belief networks is given. The simulation is based on a two steps procedure. The first one is a node deletion technique to calculate the *'a posteriori'* distribution on a variable, with the particularity that when exact computations are too costly, they are carried out in an approximate way. In the second step, the computations done in the first one are used to obtain random configurations for the variables of interest. These configurations are weighted according to the importance sampling methodology. Different particular algorithms are obtained depending on the approximation procedure used in the first step and in the way of obtaining the random configurations. In this last case, a stratified sampling technique is used, which has been adapted to be applied to very large networks without problems with round-off errors.

Keywords: Belief networks, simulation, importance sampling, stratified sampling, approximate precomputation.

1 Introduction

The problem of probability propagation is defined as the process of calculating the probability values of some variables in a dependence graph, given a set of observed variables. Many algorithms have been proposed in the last years to solve this problem in an exact way [13, 14, 17, 19, 20, 21, 24]. These methods take advantage of conditional independences among the variables given by the structure of the network, to do the propagation by local computations. Although they give exact results, all these algorithms are NP-hard [5] in the worst case. So, if the network is complicate enough, other types of algorithms should be considered.

This fact motivates the development of approximate algorithms. Most of them try to obtain a good estimation of the probabilities in the network by using Monte Carlo techniques. Although approximate inference is NP-hard too [6], the class of resolvable problems is wider.

Monte Carlo algorithms for belief networks can be classified into two different groups: those based on importance sampling and those based on Markov chains. In both cases, the problem of probability propagation may be viewed as the obtainment of samples form a difficult to manage probability distribution. Importance sampling algorithms are based in the use of a modified distribution in order to obtain independent samples, that are weighted to resemble the original distribution. The first algorithm into this group, called Probabilistic Logic Sampling, is due to Henrion [9]. It provides good results when no evidences are given. One improved algorithm, called Likelihood Weighting, was proposed by Fung and Chang [8] and Shachter and Peot [22]. It performs well, but in some cases, the same problem as in Logic Sampling may arise [3]: all the weights are 0 or 1. A more sophisticated method was proposed by Cano, Hernández and Moral, based on entropy criteria [3]. Given the conditional distributions in the network, they use those ones with less entropy (the most informative ones) to obtain the samples, and those ones with more entropy to compute the weights. This way they try to solve the problem of 0-1 weights.

In the case of Markov Chain-Monte Carlo algorithms, the samples are not independent: they verify the Markov property. The best known algorithm using this technique is Pearl's stochastic simulation [16]. This method was generalized by Jensen, Kong and Kjærulff [12], allowing samples to be generated with a greater degree of independence, but with a higher computational cost.

In this paper we study a general class of importance sampling algorithms, presented in [11]. They perform making a first approximate propagation based on the concept of node removal [24], similar to D'Ambrosio's symbolic propagation [21]. The results obtained from this first propagation are improved by sampling the obtained functions.

Bouckaert [1] and Bouckaert, Castillo and Gutiérrez [2] developed a stratified sampling scheme for probability propagation. It performs well, but when the network is too complex, precision problems make this method infeasible. Here it is considered how to modify that algorithm to use the same sampling distributions as in importance sampling. Also, a method is presented to avoid rounding errors when simulating.

In section 2, the basic notation is established and the general problem is formulated. In section 3, importance sampling is introduced. The new algorithms are treated in detail, studying several variations over the main scheme, in section 4. Section 5 is devoted to the application of stratified sampling to the new algorithms. A new way of performing stratified sampling, avoiding numerical errors, is presented in section 6, while in section 7 experimental results are commented. The paper ends with conclusions and further research in section 8.

2 Notation and Problem Formulation

A *belief network* is a directed acyclic graph where each node represents a random variable, and the topology of the graph shows the independence relations among the variables, according to the d-separation criterion [17]. Given the independences attached to the graph, the joint distribution is determined giving a probability distribution for each node conditioned to its parents.

Let $X = \{X_1, \ldots, X_n\}$ be the set of variables in the network. Assume each variable X_i takes values on a finite set U_i. We shall denote by U_I the cartesian product $\prod_{i \in I} U_i$. Given $x \in U_I$ and $J \subseteq I$, we shall denote by $x^{\downarrow J}$ the element of U_J obtained from x dropping the coordinates not in J. Given a function f defined over U_I, $s(f)$ will denote the set of indices of the variables for which f is defined (i.e. $s(f) = I$). Under these conditions, the conditional distribution of X_i given its parents in the network, $F(i)$, is denoted by

$$f_i(x) = f_i(x^{\downarrow i}, x^{\downarrow F(i)}) \quad \forall i \in N \quad \forall x \in U_{s(f_i)} \tag{1}$$

where $N = \{1, \ldots, n\}$, and

$$\sum_{x_i \in U_i} f_i(x_i, x) = 1 \quad \forall x \in U_{F(i)} \tag{2}$$

Then, the joint probability distribution for the n-dimensional random variable X can be expressed as

$$p(x) = \prod_{i \in N} f_i(x^{\downarrow s(f_i)}) \quad \forall x \in U_N \tag{3}$$

An *observation* is the knowledge about the exact value $X_i = e_i$ of a variable. The set of observations will be denoted by e, and called the *evidence set.* E will be the set of indices of the observed variables.

Every observation $X_i = e_i$ is attached to a Dirac function defined on U_i as follows:

$$\delta_{e_i}(x) = \begin{cases} 1 & \text{if} \quad e_i = x \\ 0 & \text{if} \quad e_i \neq x \end{cases} \tag{4}$$

The goal of probability propagation is to calculate the *posterior* probability function $p(x|e)$, for every $x \in U_I$, where $I \subseteq N$. This probability could be obtained from the joint distribution (3), but we assume that it is difficult to manage. Notice that it is equal to $p(x \cap e)/p(e)$, and as $p(e)$ is constant, it is proportional to $p(x \cap e)$. So, we can know the *posterior* probability if we compute for every $x \in U_I$ the value $p(x \cap e)$, normalizing afterwards. $p(x \cap e)$ can be expressed in the following way,

$$p(x \cap e) = \sum_{\substack{y^{\downarrow E} = e \\ y^{\downarrow I} = x}} \prod_{i \in N} f_i(y^{\downarrow s(f_i)}) = \sum_{y^{\downarrow I} = x} \left(\prod_{i \in N} f_i(y^{\downarrow s(f_i)}) \right) \cdot \left(\prod_{j \in E} \delta_{e_j}(y^{\downarrow j}) \right) = \sum_{y^{\downarrow I} = x} f(y) \tag{5}$$

The following concepts are used throughout the paper.

Definition 1 *Given a function f defined over a set of variables X_I and $J \subseteq I$, we define the marginalization of f over a variable X_J (or the deletion of variables in $I - J$) as a new function $f^{\downarrow J}$ defined over the set J given by the expression*

$$f^{\downarrow J}(x) = \sum_{\substack{y \in U_I \\ y^{\downarrow J} = x}} f(y) \quad \forall x \in U_J \tag{6}$$

Definition 2 *Given r-functions $f_1, \ldots, f_r$ each one defined over the sets $I_1, \ldots, I_r$, a new function, called the combination of them, is defined over the set $I = \bigcup_{i=1}^{r} I_i$ as*

$$f(x) = \bigotimes_{i=1}^{r} f_i(x^{\downarrow I_i}) = \prod_{i=1}^{r} f_i(x^{\downarrow I_i}) \quad \forall x \in U_I \tag{7}$$

Notice that the product above can be done in any ordering. Thus, combination is a commutative operator.

Definition 3 *Let f be a function defined over a set I of indices, and assume a set of variables X_J, $J \subset I$ whose values $x^{\downarrow J}$ are fixed ($x^{\downarrow J} = x_0$). The restriction of f to the values x_0 is a new function f' defined on $I - J$ according to the following expression:*

$$f'(x) = f(y) \tag{8}$$

such that $y \in U_I$, $y^{\downarrow I-J} = x^{\downarrow I-J}$ and $y^{\downarrow J} = x_0$.

Restriction can simplify the problem when observations are provided. Notice that formula (5) is still valid if we replace each function f_i by its restriction to the evidence e. This way we can work with simpler functions.

3 Importance Sampling

One method to estimate the addition in (5) is the so called importance sampling technique (this technique is described in detail in [18]). It is a Monte Carlo procedure that uses an auxiliary probability distribution $P^*(x)$, instead of $f(x)$, to obtain a sample from the space U_N.

Afterwards, a weight is assigned to each obtained configuration, $x^{(i)} \in U_N$. This weight is,

$$w_i = \frac{\left(\prod_{i\in N} f_i(x^{\downarrow s(f_i)})\right) \cdot \left(\prod_{j\in E} \delta_{e_j}(x^{\downarrow j})\right)}{P^*(x)} \tag{9}$$

It is necessary that $P^*(x) > 0$ whenever $p(x \cap e) > 0$.

Under these conditions, if $(x^{(1)}, \ldots, x^{(m)})$ is the obtained sample, an unbiased estimator of $p(x^{\downarrow I} \cap e)$ is given by

$$\xi = \frac{\sum_{x^{(i)\downarrow I} = x} w_i}{m} \tag{10}$$

It can be shown that the variance of ξ is minimized when $P^*(x)$ is proportional to $p(x \cap e)$ (see [3, 10, 18, 22]). However, selecting a probability proportional to p is not always possible, because p is difficult to handle. Notice that the variance of ξ is minimal when the weights are constant, so, the best we can do is to select P^* as close as posible to $p(\cdot|e)$.

Once P^* is selected, we can estimate $p(x \cap e)$ as follows (see [3, 22] for more details):

1. for $i = 1$ to m

(a) Generate a configuration $x^{(i)}$ according to P^*.

(b) Do

$$w_i = \frac{\prod_{i\in N} f_i(x^{\downarrow i}, x^{\downarrow F(i)})}{P^*(x)} \times \prod_{j\in E} \delta_{e_j}(x^{\downarrow j}) \tag{11}$$

2. For every $x \in U_I$

(c) Estimate $p(x \cap e)$ by using formula (10)

3. Normalize values $p(x \cap e)$

where m is the size of the sample.

4 Importance Sampling based on Approximate Computation

The most important decision when designing an importance sampling algorithm is the selection of the sampling distributon: it should be as similar as possible to the original distribution. In the particular case of a belief network, the original distribution is given as a product of conditional probability distributions. Known importance sampling algorithms [3, 8, 22] use sampling distributions close to the original conditional ones. In this way they try to obtain a sampling joint distribution close to the original joint distribution in order to get uniform weights for the configurations of the variables.

Here we propose a new approach to obtain the sampling distributions. The idea is not to use only the conditionals and likelihoods, but all the available information about a given variable. This means to use all the functions associated with the variable when we are to simulate it. This is the ideal case, but if the network is complicate enough, this process will be infeasible; concretely, the complexity of this process would be the same as in exact probability propagation, and this is what we are trying to avoid. In short, the problem is that the cost of the combination of the functions defined for a variable may be too high.

The solution we propose is to do approximate computations when the exact ones are too difficult. As we are not using the original $p(x|e)$ distribution, but an approximation P^*, we use the importance sampling technique and compute a weight for each configuration. The key point in this procedure is when and how to do approximate calculations in such a way that the computations are fast and the sampling distribution P^* as close as possible to $p(x|e)$.

The algorithm starts with a family of functions composed by the conditional distributions and the observations,

$$H = \{f_1, \ldots, f_n\} \cup \{\delta_{e_l}\}_{l \in E} \tag{12}$$

The true $p(\cdot|e)$ is proportional to the product of all the functions in H (see equation (5)).

Then, an ordering of the variables in the network is considered, given by a permutation σ over the set $\{1, \ldots, n\}$. The next step is to *delete* the variables in sequence, following the ordering imposed by σ. There are two ways of doing the *deletion* of a variable $X_{\sigma(i)}$: exact and approximate.

Exact

1. Combine all the functions which are defined for variable $X_{\sigma(i)}$, obtaining a function h_i.
2. Delete $X_{\sigma(i)}$ from the combination, h_i, by marginalizing the result to $s(h_i) - \{\sigma(i)\}$.
3. Add the result to H. Remove from H all the functions which were combined to obtain h_i.

If we are able to repeat this process for all the variables in each step we obtain a sampling distribution proportional to $p(x \cap e)$. In fact, it performs like an exact propagation algorithm [24], and the following proposition holds:

Proposition 1 *Assume the conditions for exact deletion:*

- *If h_n is the function obtained when we are deleting $X_{\sigma(n)}$ then for all $x \in U_{\sigma(n)}$, $h_n(x)$ is proportional to $p(x|e)$.*
- *If h_i is the function obtained when we are deleting $X_{\sigma(i)}$ $(i < n)$, $\Sigma(i) = \{\sigma(i+1), \ldots, \sigma(n)\}$, and $x_0 \in U_{\Sigma(i) \cap s(h_i)}$, then the restriction of h_i to x_0, h'_i (see equation (8)) is proportional to the probability $p(.|e, x_0)$: $\forall x \in U_{\sigma(i)}, h'_i(x) \propto p(x|e, x_0)$.*

Proof: First we are going to show that if H is the set of functions obtained before deleting $X_{\sigma(i)}$ $(i = 1, \ldots, n)$ then

$$\forall x \in U_{\{\sigma(i)\} \cup \Sigma(i)}, \quad p(x, e) = \prod_{h \in H} h(x^{\downarrow s(h)}) \tag{13}$$

This is trivially true for $i = 1$.

From (5) we obtain

$$\begin{aligned}\forall x \in U_{\{1,\ldots,n\}}, \quad p(x,e) &= \prod_{i=1}^{n} f_i(x^{\downarrow s(f_i)}) \prod_{j\in E} \delta_{e_j}(x^{\downarrow j}) \\ &= \prod_{h\in H} h(x^{\downarrow s(h)})\end{aligned}$$

Now if H is the set of functions obtained before deleting $X_{\sigma(k)}$ and H' the set after the deletion, we are going to show that if (13) is true for $i = k$ and H, then it is also true for $i = k+1$ and H'.

If (13) is true for $i = k$ then

$$\forall x \in U_{\{\sigma(k)\}\cup\Sigma(k)}, \quad p(x,e) = \prod_{h\in H} h(x^{\downarrow s(h)})$$

If $y \in U_{\{\sigma(k+1)\}\cup\Sigma(k+1)} = U_{\Sigma(k)}$, then

$$p(y,e) = \sum_{x^{\downarrow\Sigma(k)}=y} p(x,e) = \sum_{x^{\downarrow\Sigma(k)}=y} \left[\prod_{h\in H} h(x^{\downarrow s(h)})\right]$$

Let $H(k) = \{h \in H | \sigma(k) \in s(h)\}$, then

$$p(y,e) = \sum_{x^{\downarrow\Sigma(k)}=y} \left(\prod_{h\in H(k)} h(x^{\downarrow s(h)})\right)\left(\prod_{h\in H-H(k)} h(x^{\downarrow s(h)})\right)$$

Now, if $h \in H - H(k)$, $x^{\downarrow s(h)} = y^{\downarrow s(h)}$ and $h(x^{\downarrow s(h)}) = h(y^{\downarrow s(h)})$ not depending this value on x, for fixed y. Then

$$p(y,e) = \left[\prod_{h\in H-H(k)} h(y^{\downarrow s(h)})\right] \cdot \left[\sum_{x^{\downarrow\Sigma(k)}=y} \prod_{h\in H(k)} h(x^{\downarrow s(h)})\right]$$

Taking into account that $H' = [H - H(k)] \cup \{h_k^{\downarrow\Sigma(k)}\}$ and that

$$h_k^{\downarrow\Sigma(k)}(y) = \sum_{x^{\downarrow\Sigma(k)}=y} \left(\prod_{h\in H(k)} h(x^{\downarrow s(h)})\right)$$

then,

$$p(y,e) = \left[\prod_{h\in H-H(k)} h(y^{\downarrow s(h)})\right] \cdot h_k^{\downarrow\Sigma(k)}(y)$$

So, we obtain the desired result:

$$\forall y \in U_{\{\sigma(k+1)\}\cup\Sigma(k+1)}, \quad p(y,e) = \prod_{h\in H'} h(y^{\downarrow s(h)})$$

Now, we can use the form (13) to show the proposition for every $i = 1, \ldots, n$. In fact, we have

$$\begin{aligned}
\forall x \in U_{\sigma(i)\cup\Sigma(i)}, \quad p(x, e) &= \prod_{h \in H} h(x^{\downarrow s(h)}) \\
&= \left[\prod_{h \in H(i)} h(x^{\downarrow s(h)})\right] \cdot \left[\prod_{h \in H-H(i)} h(x^{\downarrow s(h)})\right] \\
&= h_i(x^{\downarrow s(h_i)}) \prod_{h \in H-H(i)} h(x^{\downarrow s(h)})
\end{aligned}$$

Let $x = (y, x_0, z)$, where $y \in U_{\{\sigma(i)\}}$, $z \in U_{\Sigma(i)-s(h_i)}$, $x_0 \in U_{\Sigma(i)\cap s(h_i)}$, x_0 constant. Then,

$$p(x, e) = p(y, x_0, z, e) = h_i((y, x_0)^{\downarrow s(h_i)}) \prod_{h \in H-H(i)} h((x_0, z)^{\downarrow s(h)})$$

Taking the sum on $z \in U_{\Sigma(i)-s(h_i)}$, we have

$$\begin{aligned}
p(y, x_0, e) &= \sum_{z \in U_{\Sigma(i)-s(h_i)}} p(y, x_0, z, e) \\
&= h_i((y, x_0)^{\downarrow s(h_i)}) \left(\sum_{z \in U_{\Sigma(i)-s(h_i)}} \left[\prod_{h \in H-H(i)} h((x_0, z)^{\downarrow s(h)})\right]\right)
\end{aligned}$$

For x_0 constant the second factor is constant and therefore,

$$p(y, x_0, e) = k \cdot h_i((y, x_0)^{\downarrow s(h_i)}) = k \cdot h_i'(y)$$

From this expression we obtain the desired result:

$$\forall y \in U_{\sigma(i)}, \quad p(y|x_0, e) = \frac{k}{p(x_0, e)} h_i'(y) = k' \cdot h_i'(y) \propto h_i'(y)$$

□

The two properties in the proposition above, allow us to simulate a value $x \in U_N$ with a probability equal to $p(x|e)$. What we have to do is to simulate values for the variables in the ordering $X_{\sigma(n)}, \ldots, X_{\sigma(1)}$. To obtain a value for a variable $X_{\sigma(i)}$, we sample from the function h_i, making the restriction of it to the values x_0 already obtained for the preceding variables, $X_{\Sigma(i)}$, and normalizing afterwards.

In some cases, the size [1] of h_i would be so big that its computation is infeasible. Then we have to do the deletion of the variables in an approximate way. Several approximate criteria may be defined, but always according to the following scheme:

Approximate

1. Let be $H(i) = \{h \in H \mid \sigma(i) \in s(h)\}$, the set of functions which are defined for variable $X_{\sigma(i)}$. Remove $H(i)$ from H.

2. Transform $H(i)$ by combination. To do this, we repeat several times the following process: take $R \subset H(i)$. Combine all the functions in R. Add the combination to $H(i)$. Remove R from $H(i)$.

3. Calculate $H^+(i)$ from $H(i)$ by deleting $X_{\sigma(i)}$ in all the functions belonging to $H(i)$.

4. Add $H^+(i)$ to H.

This procedure coincides with the exact one if in the second step, all the functions in $H(i)$ are combined. The idea of the approximate approach is to combine functions while a fixed size threshold is not surpassed. One important property of the approximate step is that no new 0 values are added. That is, if $x \in U_N$ is such that $h(x^{\downarrow s(h)}) \neq 0$ for every $h \in H$, before deleting $X_{\sigma(i)}$, this property is verified after the deletion of the variable.

The goal of the algorithm is to obtain configurations of the variables X_N. The process to simulate a value for a variable $X_{\sigma(i)}$ in the approximate approach is as follows: If x_0 is the obtained configuration for the variables $X_{\Sigma(i)}$, then

1. Let $H(i)$ be the set calculated in step 2 of the approximate deletion procedure.

2. Restrict each function in $H(i)$ to x_0. Combine all the functions in $H(i)$, obtaining a new function h'_i defined over $U_{\sigma(i)}$.

3. If $N(h'_i)$ is the normalization of h'_i, obtain a value for $X_{\sigma(i)}$ following the probability distribution $N(h'_i)$.

At this point we are able to formulate a general importance sampling algorithm for probability propagation:

[1] The size of a function h is defined as the product of the number of cases of all the variables for which h is defined.

The Main Algorithm (ALG IS)

```
1. Let $H = \{f_i \mid i = 1, \ldots, n\}$.
2. Select an ordering $\sigma$ for the variables in $G$.
3. Incorporate observations:
    (a) Restrict all the functions in $H$ to the evidences $\{e_l\}_{l \in E}$ according to equation (8).
    (b) For every observed variable $X_l$, $l \in E$ do $H = H \cup \{\delta_{e_l}\}$
4. for $i = 1$ to $n$ do
    (a) Delete $X_{\sigma(i)}$ by the approximate procedure[2]
5. for $j = 1$ to $m$ do
    (a) $w_j = 1.0$
    (b) for $i = n$ downto 1 do
        i. Obtain a value for $X_{\sigma(i)}$, $x_i^{(j)}$, according to $N(h'_i)$.
        ii. Do
                $w_j = \frac{w_j}{N(h'_i)(x_i^{(j)})}$
    (c) Do
            $w_j = w_j \times \prod_{i=1}^{n} f_i(x_i^{(j)}) \times \delta_{e_i}(x_i^{(j)})$
6. Estimate the desired probabilities according to equation (10).
```

At the end of loop 5.(b), the weight w_j takes the value

$$w_j = 1/P^*(x^{(j)})$$

where P^* is the sampling disribution we use, given by:

$$P^*(x^{(j)}) = \prod_{i=1}^{n} N(h'_i)(x^{(j)\downarrow i}) \tag{14}$$

After step 5.(c) the resulting weight is the correct one for importance sampling (equation (11)).

Observe that the efficiency of the algorithm depends on the ordering of the variables at step 2.

[2] Remember that the exact computation is a particular case of this step; so the exact deletion is also possible.

4.1 Particular Cases of the Main Algorithm

In this section we shall consider several variations from the main algorithm. The particular algorithms will be determined by the way in which $H^+(i)$ is calculated; that is, we must define criteria to select the subsets R from $H(i)$ of functions that will be combined before marginalizing, when a variable $X_{\Sigma(i)}$ is going to be deleted.

The first criterion one can think about is to take $R = H(i)$. This coincides with the exact deletion algorithm. We shall not take this criterion into account, because if we have been able to delete all the variables in an exact way, in a similar amount of time, the exact probabilities can be computed with no need to simulate.

As the opposite of the preceding, one may decide not to combine any function, that is, $H(i)$ does not change in the second step of the approximate scheme. This is the faster procedure we can consider, because no time is spent on computing the combinations. However, simulation time may grow if many functions are associated to the variables. In this case, when a variable is being simulated, all its attached functions have to be evaluated for the current configuration of the variables. Moreover, the estimations obtained through this method should be the worst, because we lose information in each deletion (remember that the correct way is first to combine and then marginalize).

Now we shall study three groups of methods for combining the functions.

4.1.1 Size Criteria

In this section we define three criteria attending the size of the functions resulting from the combinations.

The first idea is to combine all the functions concerning a variable if the size of the resulting function does not exceed a given threshold. That is, if the fixed size is not surpassed, the exact deletion is done; otherwise, no functions are combined. This threshold can be fixed taking into account the amount of memory avaliable in the system.

This method can be improved if we realize that maybe we cannot combine all the functions, but only some of them. The procedure would be to select R in the approximate deletion by including functions while the resulting combination does not surpass the established limit. Notice that, in this case, the ordering in which the functions are selected is important. Two approaches can be considered here:

- combine functions in an arbitrary sequence while maximum size is not exceeded (*criterion 1*), or

- combine first those that maximize the size of the common variables (*criterion 2*). Notice that functions defined over a single variable can be combined without adding complexity. Thus, unitary functions should be the first to be combined.

Example 1 *Assume variable X_i is going to be deleted, and it is associated to the following functions:*

$$f_1(X_i, X_j, X_k), f_2(X_i, X_l, X_m), f_3(X_i, X_k), f_4(X_i)$$

every variable with three possible values, and suppose the size threshold is 27. The functions remaining after the combination process would be:

criterion 1	$h_i^1 = f_1 \otimes f_3 \otimes f_4$ $h_i^2 = f_2$
criterion 2	$h_i^1 = f_2 \otimes f_3$ $h_i^2 = f_1 \otimes f_4$

Experimental evaluation shows criterion 2 to be the best of this group.

4.1.2 Entropy Criteria

We are interested in improving the results given by criterion 2. The solution could be to make a higher effort in the combination process. The approach we propose here is the following: supose after applying criterion 2, not all the functions have been combined. One way to refine the process could be to select the most ”suitable” remaining function, and combine it with that one with a higher common size (as in criterion 2). However, notice that in this case we are allowing to surpass the size threshold considered in the preceding section, but this happens at most once for each variable. In many cases, the increment of computational cost can be assumed.

The key point here is what do we mean by the most ”suitable” function. One approach can be to consider the quality of the remaining functions. Namely, the function providing the biggest amount of information should be the most likely to be combined. That amount of information can be measured through Shannon entropy, which is defined as follows:

Definition 4 *Given a probability mass function f, defined over a finite set Ω, its entropy is defined as:*

$$E(f) = -\sum_{x \in \Omega} f(x) \log f(x) \tag{15}$$

The higher the entropy of a function f is, the less informative f is. The maximum ($\log |\Omega|$) is reached when f is the uniform distribution. So, the criterion we may consider is to combine the function whose entropy is minimum. This function should be combined with that one whose domain is more similar. We refer to this as *criterion 3*. The detailed algorithm is as follows:

1. Let $X_{\sigma(i)}$ be the variable being deleted.
2. Let $H(i)$ be the set of functions resulting from the combination process according to criterion 2.
3. Select $h \in H(i)$ such that $E(h) = \max_{f \in H(i)} E(f)$
4. Remove h from $H(i)$.
5. Select $h^* \in H(i)$ such that $\|s(h) \cap s(h^*)\| = \max_{f \in H(i)} \|s(h) \cap s(f)\|$, where $\|I\|$ is the product of the number of cases of the variables whose indices are in I.
6. Remove h^* from $H(i)$.
7. $H(i)$=$H(i) \cup \{h \otimes h^*\}$

4.2 Dealing with Observations

Another important question to establish about the main algorithm is the treatment of the observed variables. In this work, we have decided to incorporate them before calculating the sampling distributions. In this way, the functions are restricted to the observed values. Thus, the size of some functions will be reduced. Moreover, there is no need to distinguish between observed and not observed variables during the simulation process. This fact reduces the possibility of obtaining null weights. Notice that if the functions are reduced to the observed values, any configuration obtained in the simulation will be consistent with the evidences. One of the causes of null weights in existing simulation algorithms is the discordance between the simulated values and the observations. The bond is that the dinamical inclusion of new evidences requieres a new computation of the sampling distributions.

The other option is not to restrict the functions to the evidences. In this case, the observed variables are not simulated; they take directly the observed value. Then, the probability of the evidence is used to weight the simulation. This is the way Likelyhood Weighting performs. The advantage is that the inclusion of new evidences is easy, but the counterpart is that in some cases all the weights result to be zero.

4.3 Selecting an Elimination Ordering

The ordering in which variables are eliminated when computing the sampling distributions determines the efficiency of the propagation algorithm. This problem is similar to the construction of optimal cluster trees in exact propagation algorithms. These algorithms requiere a previous triangulation of the moral graph associated to the causal network. Triangulation is done by deleting the variables in the network and connecting all the nodes adjacent to the node being deleted. An appropriate selection of the deletion ordering should lead to the obtainment of cliques with a lower size.

In the case of importance sampling algorithm, an appropriate ordering may result in functions with a lower size, and this way, more exact computations could be done without exceeding the size threshold.

We think that orderings resulting from triangulation processes would produce good results in our algorithms. Also, it can be considered to remove, at each step, the variable producing the smallest function. In the experimentation carried out in this paper, we have considered a deletion ordering from leaves to roots. So, the variables will be simulated from roots to leaves. This is the simulation ordering used in the Likelihood Weighting algorithm. The following proposition establishes when our algorithms and Likelihood Weighting are equivalent.

Proposition 2 *Let $X = \{X_1, \ldots, X_n\}$ be the set of variables in a causal network, and $H = \{f_1, \ldots, f_n\}$ the conditional probabilities for each variable. Assume there are no observations. Then, if σ is an ancestral ordering [3] of the vertices in the network, and the deletion sequence is from $\sigma(n)$ to $\sigma(1)$, importance sampling algorithm is equivalent to likelihood weighting.*

Proof: It is sufficient to prove that for each variable X_i, $1 \leq i \leq n$, its sampling distribution is just f_i, i.e. the conditional distribution for X_i given its parents in the network. It is clear that any variable $X_{\sigma(i)}$ never appears in functions $f_{\sigma(j)}$, $j < i$, since σ is an ancestral ordering.

Also, given that σ is an ancestral ordering, the first variables to be removed are those being leaves in the network. Now, assume X_i is a leaf. Its only attached distribution is $f_i(x^{\downarrow i}, x^{\downarrow F(i)})$, $\forall x \in U_{s(f_i)}$. After removing X_i, the remaining function is

$$\sum_{x_i \in U_i} f_i(x_i, x), \quad \forall x \in U_{F(i)}$$

and, by equation (2) it is equal to 1.

[3] An ordering of the nodes in a graph is said to be ancestral if any descendant of any given node appears after it in the ordering sequence.

Now assume all the leaves have been removed. At this point, the next variables to be removed are those whose only descendants were leaves. Thus, themselves are leaves now, and their associated distributions are the conditionals combined with those resulting from the deletion of the previous variables, but we have seen the result of the deletion to be 1. So, the sampling distribution for each variable is the conditional distribution of that variable given its parents. □

5 Applying Stratified Sampling to the New Algorithms

In this section, we apply the stratified sampling technique [2, 18] to the new algorithms proposed in this paper. Stratified sampling is a well known simulation technique in Statistics. The basic idea is to divide the sample space into several regions (also called *strata*) and then choose an optimum number of configurations from each region. In this way, rare samples may be avoided. Theoretical basis for stratified sampling can be found in [18].

5.1 Stratified Sampling in Belief Networks

First propagation algorithms based on stratified sampling were proposed by Bouckaert [1] and Bouckaert, Castillo and Gutiérrez [2]. The idea is to consider the space of all possible configurations of the variables in the network, so that most probable configurations are assigned to a greater region. Then, configurations are selected seeking over the whole region. The procedure is as follows:

Let $X = \{X_1, \ldots, X_n\}$ be a set of discrete random variables, each variable X_i taking values on a set $\{0, 1, \ldots, r_i - 1\}$. Let $f_i(x^{\downarrow i}, x^{\downarrow F(i)})$, $i = 1, \ldots, n$ be the conditional distributions for each variable given its parents in the network. Under these conditions we can generate all the possible configurations from the variables and compute their associated probabilities. The following ordering of the configurations is defined [2]:

Definition 5 *Let $I_1 = (x_1, x_2, \ldots, x_n)$ e $I_2 = (y_1, y_2, \ldots, y_n)$ be two configurations of X. I_1 precedes I_2 ($I_1 < I_2$) if and only if:*

$$I_1 < I_2 \Leftrightarrow \exists k \text{ such that } \forall j < k \quad x_j = y_j \text{ and } x_k < y_k \tag{16}$$

Based on the ordering in (16), a table is constructed representing the sample space. This table is used to seek for the configurations. For example, let $X = (X_1, X_2)$ be a set of variables, each one with two possible values. Figure 1 shows the network associated to X and the table 1 shows the *'a priori'* probabilities for the variables in X. In table 2 we can find the sorted

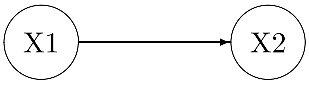

Figure 1: A belief network.

X_1	X_2
$f_1(X_1 = 0) = 0.6$	$f_2(X_2 = 0, X_1 = 0) = 0.2$
$f_1(X_1 = 1) = 0.4$	$f_2(X_2 = 0, X_1 = 1) = 0.5$
	$f_2(X_2 = 1, X_1 = 0) = 0.8$
	$f_2(X_2 = 1, X_1 = 1) = 0.5$

Table 1: Probability values for the sample net.

Configuration	Probability	Cumulative prob.	Associated interval
(0,0)	0.12	0.12	(0.00,0.12)
(0,1)	0.48	0.60	(0.12,0.60)
(1,0)	0.20	0.80	(0.60,0.80)
(1,1)	0.20	1.00	(0.80,1.00)

Table 2: Probabilities and intervals for the sorted configurations.

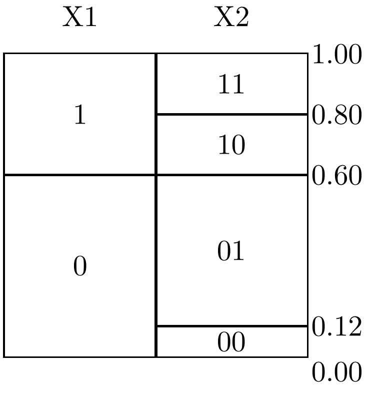

Figure 2: Configurations and their cumulative probabilities.

configurations with their probability of occurence, cumulative probability and associate interval. Each configuration is attached to an interval $I_i = [l(i), h(i)) \subseteq [0,1]$ whose limits are computed according to the following expressions:

$$\begin{aligned} l(i) &= \sum_{j<i} \prod_{r=1}^{n} f_r^*(I_j^{\downarrow r}) \\ h(i) &= l(i) + \prod_{r=1}^{n} f_r^*(I_j^{\downarrow r}) \end{aligned} \tag{17}$$

where I_j is the j-th configuration and f_r^*, $r = 1, \ldots, n$, are the sampling distributions. Figure 2 shows the partition for the network in figure 1 over the interval $[0,1]$.

To obtain a sample of size m, it performs generating m numbers into $[0,1]$, and retrieve the configuration correspondig to each number according to the partition of the region (figure 2). Then, each configuration is scored according to the distribution used to compute the intervals (f_r^*) and the original distribution. The m numbers are not random, but they are calculated in a deterministic way as [2],

$$k_i = \frac{i - 0.5}{m} \quad i = 1, 2, \ldots, m$$

Example 2 *Consider the net in figure 1. Generating four numbers $k_i = (i-0.5)/4$, $i = 1, \ldots, 4$, we obtain the sequence,*

$$(0.125, 0.375, 0.625, 0.875)$$

and its associated sample is,

$$(01), (01), (10), (11)$$

Notice that when m grows, the relative frequency for each configuration converges to its probability value. Several sampling distributions can be used. Bouckaert, Castillo and Gutiérrez [2] use the same functions as in Likelihood Weighting algorithm. We use the functions developed in section 4.

5.2 The Stratified Sampling Algorithm

In this section we formalize the method explained in previous section. The question to solve is how to obtain the configuration associated to a given number $k_i \in [0,1]$. One procedure could be to compute the cumulative probabilities for all the configurations in sequence until k_i is reached. However, this is the same as computing the exact probabilities, what is precisely what we are trying to avoid.

The method proposed by Bouckaert et al. starts with a configuration of the variables, and, for a given k_i, determines what variables must change their value to reach k_i. For this purpose, associated to each variable there will be an interval $[l(i), h(i))$ representing the "zones" of the region in which the variable X_i changes its value (see figure (2)). Then, it searches for the first variable X_j such that $k_j \in [l(j), h(j))$, starting from $j = n$. When j is found, the intervals corresponding to the variables $X_{j+1}, \ldots, X_n$ are updated for them to contain k_i.

One important advantage of stratified sampling is that it is possible to know how many values from the sequence k_i correspond to the same configuration. For a given $k \in [0, 1]$, this is achieved by [2],

$$\Delta = \lfloor (h(n) - k_i) \cdot m \rfloor + 1 \tag{18}$$

where $\lfloor x \rfloor$ is the integer part of x for all $x \in I\!\!R$.

The general algorithm is as follows:

The Stratified Algorithm (ALG SS)

1. Select an ordering σ for the set of indices $N = \{1, \ldots, n\}$.
2. Compute the sampling distributions f_l^*, $l \in N$ according to the deletion sequence imposed by σ (as in importance sampling).
3. Initialization.
4. For $j = 1$ to m do
   (a) Generate configuration $x^{(j)}$
   (b) Compute
   $$\Delta = \lfloor (h(n) - k_j) \cdot m \rfloor + 1$$
   (c) Compute
   $$w_j = \Delta \cdot \frac{\left[\prod_{i \in N} f_i(x^{(j) \downarrow s(f_i)})\right] \left[\prod_{r \in E} \delta_{e_r}(x^{(j) \downarrow r})\right]}{\prod_{l \in N} f_l^*(x^{(j) \downarrow s(f_l^*)})}$$
   (d) $j = j + \Delta$
5. Estimate the probabilities $p(x \cap e)$ according to equation (10).

where m is the sample size.

This algorithm is very similar to importance sampling. The differences remain in inicialization and configurations generation procedures. These procedures are as follows [2]:

Initialization

```
1. $l(0) = 0.0;\ \ h(0) = 1.0$

2. For $i = 1$ to $n$ do

   (a) $l(i) = 0.0$
   (b) If $X_{\sigma(i)} \in X_E$ then
         • $val(\sigma(i)) = e_{\sigma(i)}$
         • $h(i) = h(i-1)$
       else
         • $val(\sigma(i)) = 0$
         • $h(i) = h(i-1) \cdot f^*_{\sigma(i)}(X_{\sigma(i)} = 0)$
```

This procedure computes the first configuration and its probability. The configuration is stored in array $val(.)$. Also, intervals $[l(i), h(i))$ are initialized according to the probability of the first value of each variable X_i.

The procedure to generate configurations is,

Generate configuration $x^{(i)}$

```
1. $k_i = \frac{i - 0.5}{m}$

2. If $l(n) \leq k_i \leq h(n)$ then $j = n$
   else select $j < n$ such that $h(j-1) \geq k_i \geq h(j)$

3. While $j \leq n$ do
   If $X_{\sigma(j)} \in X_E$ then
     • $l(j) = l(j-1)$
     • $h(j) = h(j-1)$
   else
     • $t = 0$
     • $l(j) = l(j-1)$
```

```
        • $h(j) = l(j) + (h(j-1) - l(j-1)) \cdot f^*_{\sigma(j)}(X_{\sigma(j)} = t)$
        • While $k_i > h(j)$ do
            – $t = t + 1$
            – $l(j) = h(j)$
            – $h(j) = l(j) + (h(j-1) - l(j-1)) \cdot f^*_{\sigma(j)}(X_{\sigma(j)} = t)$
        • $val(\sigma(j)) = t$
      $j = j + 1$
  4. Return configuration $x^{(i)}$ stored in array $val(.)$.
```

k_i is the point into interval $[0, 1]$ for which we want to know the associate configuration of the variables. At step (2), it is computed the position from which the configuration of the variables have to be updated. This position corresponds to the first variable for which its associated interval $[l, h)$ contains k_i. At step (3) the configuration is obtained and intervals are updated. Notice that observed variables do not influence the construction of the intervals.

5.3 Problems of Stratified Sampling

Although stratified sampling is a very efficient algorithm, in practice, a very important problem arises when we deal with large networks. The problem is due to the limited precision of the real numbers into the computers. Notice that when computing the intervals $[l, h)$, we use the following expression,

$$h(j) = l(j) + (h(j-1) - l(j-1)) \cdot f^*_{\sigma(j)}(X_{\sigma(j)} = t)$$

So, we are performing two additions and one product with numbers very close to zero. In general, the larger the network is, the smaller the numbers are. The result is that if we use single floating point numbers to represent the limits of the intervals, they all are rounded to zero, because of the loss of significant digits. Hence, the algorithm is unable to find any configuration for the variables.

The solution could be to increase the number of bits used to represent floating point numbers into the computer, but we always can find a large enough network so that all the intervals are rounded to zero.

In the following section we propose an alternative method for doing stratified sampling avoiding precision problems.

6 Recursive Stratified Sampling

Whether stratified sampling works or not, depends strongly on the size of the network. It would be interesting to find a way in which this method could be applied to any network independently of its size. Here we propose a propagation scheme where stratified sampling is applied to each variable separately. In each step, the intervals are scaled to $[0, 1]$; hence, no rounding errors appear. We will start with a first rough approach to the algorithm, an example of how it works and finally the detailed algorithm, which can be implemented in a recursive way. A first approach may be like this:

1. Select an ordering $\{X_1, \ldots, X_n\}$ for the variables in the network and compute the sampling distributions.
2. Take m numbers $k_i \in [0, 1]$ as in original stratified sampling.
3. For $t = 1$ to $n - 1$ do
    (a) Consider X_t. (Initially $t = 1$).
    (b) For each possible value of X_t do
        i. Let r be the amount of numbers k_i laying into the interval corresponding to the current value of X_t.
        ii. Generate the r configurations of the variable X_{t+1}.

When the procedure is finished, each variable has beed simulated m times. The next example is meant to clarify the idea.

Example 3 *Consider the network in figure 1, and the same sequence of numbers as in example 2. When we generate the four numbers k_i, we can know how many numbers k_i are in the interval associated with the first case of the variable X_1. In this case, in interval $[0, 0.6)$ there are two numbers: $k_1 = 0.125$ and $k_2 = 0.375$. So, we know that in the final sample there will be two configurations (x_1, x_2) with $x_1 = 0$.*

If we use Bouckaert's stratified sampling, then we should work on the interval $[0, 0.6)$ in order to obtain the configurations of X_2 for the values k_1 and k_2. That is, we should apply (17). But, we want remove the problem of rounding. For this task, we scale the interval $[0, 0.6)$ in the interval $[0, 1)$, and the values k_1 and k_2 are changed to $k'_1 = 0.2083$ and $k'_2 = 0.625$.

Then, we repeat the process for X_2, for which we have to generate two values, but now working in the interval $[0, 1]$. In this case, in the new interval $[0, 0.2)$ there are not any k'_i for the first

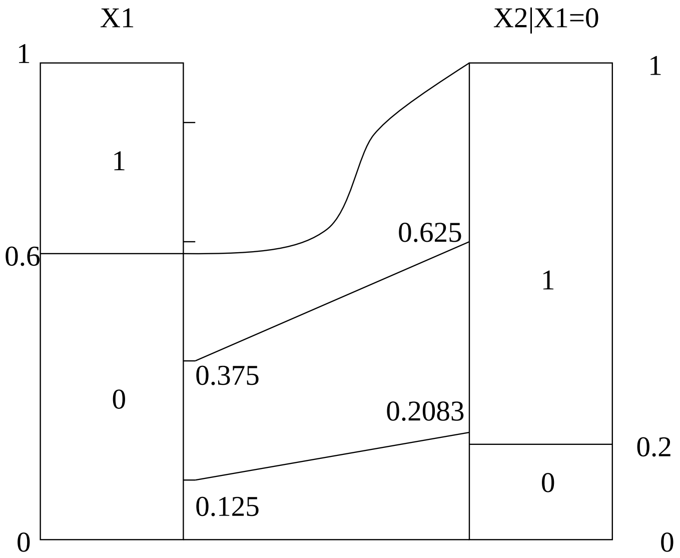

Figure 3: Recursive stratified sampling.

configuration of X_2. And for the second configuration of X_2, and in the new interval $[0.2, 1)$, there are two k'_i. So, we obtain two configurations (x_1, x_2) with $x_2 = 1$, the ones associated with k'_1 and k'_2.

Since we have no more variables, then we come back to the first variable in order to resamble the whole configurations. In this case, as k'_1 and k'_2 are really the values k_1 and k_2 in the previous interval, then we can finish saying that there are two configurations of the manner $(0, 1)$ in the final sample.

So we obtain the same result that in the usual stratified sampling but removing rounding errors.

At this point, we can give a detailed version of the algorithm, expressed as a recursive procedure.

Recursive Stratified Algorithm (ALG RSS)

1. Select an ordering σ for the set of indices $N = \{1, \ldots, n\}$.

2. Compute the sampling distributions f_l^*, $l \in N$ according to the deletion sequence imposed by σ (as in importance sampling).

3. Let m be the sample size.

4. Increment: $inc = \dfrac{1}{m}$

```
5. First number: $k = \frac{0.5}{m}$
6. Interval limits: $l = 0.0$, $h = 1.0$
7. Variable to be simulated: $i = 1$ (the first one).
8. SIMULATE($i$,$l$,$h$,$k$,$inc$,$m$)
9. Estimate probabilities $p(x \cap e)$ according to equation (10), using the weights computed in
   procedure SIMULATE.
```

where SIMULATE is a procedure performing like a classical stratified sampling just for one variable at a time, and then calling itself recursively to simulate remaining variables.

SIMULATE(i,l,h,$k^{(i-1)}$,inc,Δ)

```
1. If $i > n$ Then
    • Assign a weight to the configuration stored in array $val$ as in ALG SS step 4.(c).
    • return.
2. If $X_{\sigma(i)} \in X_E$ Then
    (a) $val[\sigma(i)] = e_{\sigma(i)}$
    (b) $k^{(i)} = k^{(i-1)}$
    (c) SIMULATE($i+1$,$l$,$h$,$k^{(i)}$,$inc$,$\Delta$)
    (d) return.
3. $k^{(i)} = \frac{k^{(i-1)} - l}{h - l}$;   $inc = \frac{inc}{h - l}$
4. $l = 0.0$;   $h = 0.0$
5. $v = 0$
6. While $k^{(i)} < 1.0$ do
    (a) $l = h$
    (b) $h = h + f^*_{\sigma(i)}(X_{\sigma(i)} = v)$
    (c) While $h < k^{(i)}$ do
```

```
        • $v = v + 1$
        • $l = h$
        • $h = h + f^*_{\sigma(i)}(X_{\sigma(i)} = v)$
    (d) $val[\sigma(i)] = v$
    (e) $\Delta = \left\lfloor \frac{h - k^{(i)}}{inc} \right\rfloor + 1$
    (f) SIMULATE($i+1$,$l$,$h$,$k^{(i)}$,$inc$,$\Delta$)
    (g) $k^{(i)} = k^{(i)} + \Delta \cdot inc$
    (h) $v = v + 1$
```

We explain briefly the procedure SIMULATE. When this procedure is called, it is checked whether all the variables have been simulated. If the answer is affirmative then we have a whole configuration, whose weight can be computed as in ALG SS step 4.(c).

Otherwise, we check whether the input variable $X_{\sigma(i)}$ is an observation or not. In the first case we do not simulate: the algorithm takes directly the observed value and the parameters remain unchanged and the next variable $X_{\sigma(i+1)}$ is simulated. This task is done in step 2.

But if the input variable is not an observation we have to select a value v for it. The third step scales the values of $k^{(i)}$ and inc to the interval $[0, 1]$. That is, we must divide the values by the amplitude of the interval for the former variable. Remember that $k^{(i)}$ is the number that will determine which value of the variable will be selected, according to the cumulative probabilities of such values, and inc is the increment to reach the next value of $k^{(i)}$.

The interval limits have to be scaled to $[0, 1]$ too, and this is determined implicitly by the condition $k^{(i)} < 1$ in step 6. In this step, we pick the lower $k^{(i)}$ and choose a value v for the variable $X_{\sigma(i)}$, later we pick the next $k^{(i)}$ and the process is repeated, and so on until we take all the values $k^{(i)}$ for the interval $[0, 1]$ associated to the variable $X_{\sigma(i)}$.

In order to select a value v for the varible we must fix the intervals

$$[l = \sum_{u<v} f^*_{\sigma(i)}(X_{\sigma(i)} = u), h = l + f^*_{\sigma(i)}(X_{\sigma(i)} = v)]$$

which contain the values $k^{(i)}$. So, given $k^{(i)}$, we must establish its associated interval $[l, h]$ to obtain a value v. This is the aim of the steps 6.(a)-6.(d).

When a interval is fixed then we obtain, in step 6.(e), the number of configurations, Δ, that contain the current case v for $X_{\sigma(i)}$. Then, in the step 6.(g), we choose the next value $k^{(i)}$ and the process is repeated. Morever, we must stablish the configurations for the the selected cases v and, as we know there are Δ, then Δ configurations of the following variables should be obtained. This is done by calling again to procedure SIMULATE in step 6.(f).

Notice that all the variables are simulated in the interval $[0,1]$, not depending on the size of the network, what makes this stratified algorithm valid for any arbitrary network.

Another advantage of this procedure with respect to classical stratified sampling, is that it is avoided to look for the current configuration, concretely, step 2 in the **Generate configuration** procedure of the stratified algorithm (section 5.2) is not necessary.

The following results show RSS being equivalent to SS.

Lemma 3 *Let $X = \{X_{\sigma(1)}, \ldots, X_{\sigma(n)}\}$ be the set of variables in the network, each variable $X_{\sigma(i)}$ taking values on a set $U_{\sigma(i)}$. Let be $N = \{1, \ldots, n\}$, and $x_0 \in U_N$ a configuration of the variables in X. If $f^*_{\sigma(j)}$, $j = 1, \ldots, n$ are the sampling distributions for each variable, then, for all $i = 1, \ldots, n$ it holds that*

$$\sum_{\substack{y<x_0^{\downarrow I}\\ y\in U_I}} \prod_{j\in I} f_j^*(y^{\downarrow j}) = \left(\prod_{j\in I'} f_j^*(x_0^{\downarrow j})\right)\left(\sum_{\substack{y<x_0^{\downarrow \sigma(i)}\\ y\in U_{\sigma(i)}}} f_j^*(y)\right) + \sum_{\substack{y<x_0^{\downarrow I'}\\ y\in U_{I'}}} \prod_{j\in I'} f_j^*(y^{\downarrow j}) \tag{19}$$

where $I = \{\sigma(1), \ldots, \sigma(i)\}$ and $I' = I - \{\sigma(i)\}$.

Proof: If we define the following events:

$$\begin{array}{rcl}
A & = & \text{“a configuration } y \in U_I \text{ is less than } x_0^{\downarrow I}\text{”} \\
B & = & \text{“a configuration } y \in U_{I-\{\sigma(i)\}} \text{ is equal to } x_0^{\downarrow I-\{\sigma(i)\}}\text{”} \\
C & = & \text{“a configuration } y \in U_{\{\sigma(i)\}} \text{ is less than } x_0^{\downarrow \sigma(i)}\text{”} \\
D & = & \text{“a configuration } y \in U_{I-\{\sigma(i)\}} \text{ is less than } x_0^{\downarrow I-\{\sigma(i)\}}\text{”}
\end{array}$$

then it holds that $A = (B \wedge C) \vee D$ and, since $A = (B \wedge C)$ and D are disjoint events, and $P^*(B \wedge C) = P^*(B) \cdot P^*(C|B)$,

$$P^*(A) = P^*(B) \cdot P^*(C|B) + P^*(D)$$

Where P^* is the sampling probability distribution. This equality is trivially true and equivalent to equation (19). □

Proposition 4 *Assume $k \in [0,1]$. Let $k^{(i)}$ be the transformed value of k when simulating variable $X_{\sigma(i)}$ in algorithm RSS. Let be $I = \{\sigma(1), \ldots, \sigma(i)\}$, $I' = I - \{\sigma(i)\}$ and $x_0 \in U_N$ a* *configuration for all the variables in the network. Then,*

$$k = \sum_{\substack{y < x_0^{\downarrow I'} \\ y \in U_{I'}}} \prod_{j \in I'} f_j^*(y^{\downarrow j}) + k^{(i)} \prod_{j \in I'} f_j^*(x_0^{\downarrow j})$$

Proof: Notice that observed variables are not simulated, and thus, they do not change the interval limits l and h.

We will do the proof by induction.

Initially, in the first run of procedure SIMULATE, $l = 0.0$ and $h = 1.0$, hence $k^{(1)} = \dfrac{k - l}{h - l} = k$, and by step 3 in the next call to procedure SIMULATE, with parameters $(2, l, h, k^{(1)}, inc, \Delta)$,

$$k^{(2)} = \frac{k^{(1)} - l}{h - l} = \frac{k^{(1)} - \sum_{\substack{y < x_0^{\downarrow \sigma(1)} \\ y \in U_{\sigma(1)}}} f_{\sigma(1)}^*(y^{\downarrow \sigma(1)})}{f_{\sigma(1)}^*(x_0^{\downarrow \sigma(1)})}$$

what follows from the fact that, for any variable $X_{\sigma(i)}$, the values of l and h are those computed before calling procedure SIMULATE when simulating the former variable:

$$l = \sum_{\substack{y < x_0^{\downarrow \sigma(i-1)} \\ y \in U_{\sigma(i-1)}}} f_{\sigma(i-1)}^*(y^{\downarrow \sigma(i-1)})$$

$$h = \sum_{\substack{y \leq x_0^{\downarrow \sigma(i-1)} \\ y \in U_{\sigma(i-1)}}} f_{\sigma(i-1)}^*(y^{\downarrow \sigma(i-1)})$$

Hence

$$k = \sum_{\substack{y < x_0^{\downarrow \sigma(1)} \\ y \in U_{\sigma(1)}}} f_{\sigma(1)}^*(y^{\downarrow \sigma(1)}) + k^{(2)} \cdot f_{\sigma(1)}^*(x_0^{\downarrow \sigma(1)})$$

Thus, the proposition holds for $i = 1$ and $i = 2$.

Now we will prove that if the proposition holds for any i, then it also holds for $i + 1$:

$$k = \sum_{\substack{y < x_0^{\downarrow I'} \\ y \in U_{I'}}} \prod_{j \in I'} f_j^*(y^{\downarrow j}) + k^{(i)} \prod_{j \in I'} f_j^*(x_0^{\downarrow j})$$

It follows from procedure SIMULATE$(i + 1, l, h, k^{(i+1)}, inc, \Delta)$, step 3, that

$$k^{(i+1)} = \frac{k^{(i)} - \sum_{\substack{y < x_0^{\downarrow \sigma(i)} \\ y \in U_{\sigma(i)}}} f^*_{\sigma(i)}(y^{\downarrow \sigma(i)})}{f^*_{\sigma(i)}(x_0^{\downarrow \sigma(i)})}$$

Thus,

$$k = \sum_{\substack{y < x_0^{\downarrow I'} \\ y \in U_{I'}}} \prod_{j \in I'} f^*_j(y^{\downarrow j}) + \left(k^{(i+1)} \cdot f^*_{\sigma(i)}(x_0^{\downarrow \sigma(i)}) + \sum_{\substack{y < x_0^{\downarrow \sigma(i)} \\ y \in U_{\sigma(i)}}} f^*_{\sigma(i)}(y^{\downarrow \sigma(i)}) \right) \prod_{j \in I'} f^*_j(x_0^{\downarrow j}) =$$

$$= \sum_{\substack{y < x_0^{\downarrow I'} \\ y \in U_{I'}}} \prod_{j \in I'} f^*_j(y^{\downarrow j}) + k^{(i+1)} \prod_{j \in I} f^*_j(x_0^{\downarrow j}) + \prod_{j \in I'} f^*_j(x_0^{\downarrow j}) \sum_{\substack{y < x_0^{\downarrow \sigma(i)} \\ y \in U_{\sigma(i)}}} f^*_{\sigma(i)}(y^{\downarrow \sigma(i)})$$

and according to lemma 3,

$$k = \sum_{\substack{y < x_0^{\downarrow I} \\ y \in U_I}} \prod_{j \in I} f^*_j(y^{\downarrow j}) + k^{(i+1)} \prod_{j \in I} f^*_j(x_0^{\downarrow j})$$

□

Theorem 5 *Let k_j, $j \in \{1, \ldots, m\}$ be a sequence of numbers in $[0, 1]$. Let $\{x^{(1)}, \ldots, x^{(m)}\}$ be the set of configurations obtained by algorithm SS for the sequence above, and $\{y^{(1)}, \ldots, y^{(m)}\}$ those ones obtained by algorithm RSS. Then,*

$$\forall i \in \{1, \ldots, m\}, \quad x^{(i)} = y^{(i)}$$

Proof: Let I and I' be as in proposition 4. For a given $k \in [0, 1]$, the configuration obtained by algorithm **SS** is that configuration x_0 verifying

$$\sum_{\substack{y < x_0^{\downarrow I} \\ y \in U_I}} \prod_{j \in I} f^*_j(y^{\downarrow j}) < k \leq \sum_{\substack{y \leq x_0^{\downarrow I} \\ y \in U_I}} \prod_{j \in I} f^*_j(y^{\downarrow j})$$

while algorithm **RSS** returns that configuration x_0 verifying that, for all $i \in \{1, \ldots, n\}$,

$$\sum_{\substack{y < x_0^{\downarrow \sigma(i)} \\ y \in U_{\sigma(i)}}} f^*_{\sigma(i)}(y) < k^{(i)} \leq \sum_{\substack{y \leq x_0^{\downarrow \sigma(i)} \\ y \in U_{\sigma(i)}}} f^*_{\sigma(i)}(y)$$

So, to show that the theorem is true, it is sufficient to prove the following equivalence:

$$\sum_{\substack{y<x_0^{\downarrow\sigma(i)}\\ y\in U_{\sigma(i)}}} f^*_{\sigma(i)}(y) < k^{(i)} \leq \sum_{\substack{y\leq x_0^{\downarrow\sigma(i)}\\ y\in U_{\sigma(i)}}} f^*_{\sigma(i)}(y) \Leftrightarrow \sum_{\substack{y<x_0^{\downarrow I}\\ y\in U_I}} \prod_{j\in I} f^*_j(y^{\downarrow j}) < k \leq \sum_{\substack{y\leq x_0^{\downarrow I}\\ y\in U_I}} \prod_{j\in I} f^*_j(y^{\downarrow j})$$

By proposition 4, this last inequality is true if and only if

$$\sum_{\substack{y<x_0^{\downarrow I}\\ y\in U_I}} \prod_{j\in I} f^*_j(y^{\downarrow j}) < \sum_{\substack{y<x_0^{\downarrow I'}\\ y\in U_{I'}}} \prod_{j\in I'} f^*_j(y^{\downarrow j}) + k^{(i)} \prod_{j\in I'} f^*_j(x_0^{\downarrow j}) \leq \sum_{\substack{y\leq x_0^{\downarrow I}\\ y\in U_I}} \prod_{j\in I} f^*_j(y^{\downarrow j})$$

and by lemma 3, this is equivalent to

$$\sum_{\substack{y<x_0^{\downarrow\sigma(i)}\\ y\in U_{\sigma(i)}}} f^*_{\sigma(i)}(y) < k^{(i)} \leq \sum_{\substack{y\leq x_0^{\downarrow\sigma(i)}\\ y\in U_{\sigma(i)}}} f^*_{\sigma(i)}(y)$$

□

7 Experimental Evaluation

This section presents the results of the empirical analysis carried out to test the performance of the new algorithms. They have been used to propagate in a 50 variables network, with both dense and sparse zones. We have done three different experiments. In the first one, the probability distributions have been generated following an uniform distribution $\mathcal{U}(0,1)$, and no observations have been considered. In the second one, the only difference is that 7 variables have been instanciated to their first value. In the third experiment, the same 7 variables have been observed, and the conditional distributions have been modified in this way: the probability of obtaining the observed value has been set to 0 except for one configuration of the parents, for which it is 1. This is to make likelihood weighting and likelihood stratified sampling fail (see [11]).

The following algorithms have been tested:

LW	Likelihood Weighting
IS2	Importance Sampling, criterion 2
IS3	Importance Sampling, criterion 3
LSS	Likelihood Stratified Sampling
SS2	Recursive Stratified Sampling, criterion 2
SS3	Recursive Stratified Sampling, criterion 3

In the step of combining the functions to obtain the sampling distributions, the threshold has been set to 512 values, that is, combinations can be done while the resulting function has no more than 512 values. The sampling ordering considered is from roots to leaves. Each algorithm has been run 100 times, except the stratified ones, given their deterministic character. Mean time and error have been computed. The number of simulations have been set from 1.000 to 10.000, in a frequency of 1.000.

For one variable X_l, the goodness of the estimation is measured as [7]:

$$G(X_l) = \sqrt{\frac{1}{|U_l|} \sum_{a_l \in U_l} \frac{(p'(a_l|e) - p(a_l|e))^2}{p(a_l|e)(1 - p(a_l|e))}} \tag{20}$$

where $p(a_l|E)$ is the true *a posteriori* probability, $p'(a_l|E)$ is the estimated value and $|U_l|$ is the number of cases of variable X_l. For a set of variables $(X_i)_{i \in I}$, the goodness of the estimation is:

$$G((X_i)_{i \in I}) = \sqrt{\sum_{i \in I} G(X_i)^2} \tag{21}$$

The experiments have been carried out in an Intel Pentium 75MHz computer, with 16 MB of RAM. The operating system was Linux 1.2.13, and the programs were implemented in C++ language.

Figures 4 to 17 show the results of the experiments. For each one, each importance sampling algorithm is compared with its stratified version, and also, importance and stratified are compared separately.

Attending the obtained results, the following can be said:

- There are no important differences neither among importance sampling algorithms nor among stratified algorithms when no evidences are given (experiment 1). It can be observed that, in general, applying stratified sampling is always advantageous.

- In experiment 2, importance sampling algorithms perform clearly better than likelihood weighting, due to the observed variables. Also, new stratified algorithms improve likelihood stratified scheme.

- The extreme case is experiment 3. Here, likelihood weighting and likelihood stratified are not able to give any estimation of the probabilities, because all weights result to be zero.

8 Conclusions

In this paper, we have revisited and expanded the class of importance sampling algorithms presented in [11], applying them to stratified schemes. An important feature is that it can be detected when exact computation is feasible, namely, when all functions can be combined in an exact way. Experimental results show the new algorithms being less sensitive to close to zero probabilities than Likelihood Weighting and classical Stratified Sampling. However, as we are dealing with a NP-hard problem, several examples where our algorithms do not work can be found. Nevertheless, we think the methodology proposed here will lead to solve a wide range of problems.

About recursive stratified sampling, it is able to work with arbitrary large networks. It is an important advance. Also it is shown to be equivalent to classical stratified sampling, thus, verifying the same theoretical convergence results as is [2].

Now we are developing a new way of approximating the sampling distributions, using decision trees to represent them. We expect this will lead to a better representation of difficult problems, namely, those where many close to zero probabilities are provided.

References

[1] Bouckaert, R.R., A stratified simulation scheme for inference in Bayesian belief networks. In *Uncertainty in Artificial Intelligence, Proceedings of the Tenth Conference*, 110-117, 1994.

[2] Bouckaert, R.R., Castillo, E., Gutiérrez, J.M., A modified simulation scheme for inference in bayesian networks. *International Journal of Approximate Reasoning* 14, 55-80, 1996.

[3] Cano, J.E., Hernández, L.D., Moral, S., Importance sampling algorithms for the propagation of probabilities in belief networks. *International Journal of Approximate Reasoning* 15, 77-92, 1996.

[4] Cooper, G.F., An algorithm for computing probabilistic propositions. In: *Uncertainty in Artificial Intelligence, 3* (L.N. Kanal, T.S. Levitt, J.F. Lemmer, eds.) North-Holland (Amsterdam) 3-14, 1989.

[5] Cooper, G.F., The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42, 393-405, 1990.

[6] Dagum, P., Luby, M., Approximating probabilistic inference in Bayesian networks is NP-hard. *Artificial Intelligence* 60, 141-153, 1993.

[7] Fertig, K.W., Mann, N.R., An accurate approximation to the sampling distribution of the studentized extreme-valued statistic. *Technometrics* 22, 83-90, 1980.

[8] Fung, R., Chang, K.C., Weighting and integrating evidence for stochastic simulation in Bayesian networks. In: *Uncertainty in Artificial Intelligence, 5* (M. Henrion, R.D. Shachter, L.N. Kanal, J.F. Lemmer, eds.) North-Holland (Amsterdam) 209-220, 1990.

[9] Henrion, M., Propagating uncertainty by logic sampling in Bayes' networks. In: *Uncertainty in Artificial Intelligence, 2* (J.F. Lemmer, L.N. Kanal, eds.) Norht-Holland (Amsterdam) 317-324, 1988.

[10] Hernández, L.D., Moral, S., Mixing exact and importance sampling propagation algorithms in dependence graphs. To appear in: *International Journal of Intelligent Systems.*

[11] Hernández, L.D., Moral, S., Salmerón, A., Importance sampling algorithms for belief networks based on approximate computation. Proceedings of the Sixth International Conference IPMU'96. Vol II, 859-864, 1996.

[12] Jensen, C.S., Kong, A., Kjærulff, U., Blocking Gibbs sampling in very large probabilistic expert systems. *International Journal of Human-Computer Studies* 42, 647-666, 1995.

[13] Jensen, F.V., Lauritzen, S.L., Olesen, K.G., Bayesian updating in causal probabilistics networks by local computation. *Computational Statistics Quarterly*, 4, 269-282, 1990.

[14] Lauritzen, S.L., Spiegelhalter, D.J., Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B* 50, 157-224, 1988.

[15] Monti, S., Cooper, G.F., Bounded recursive approximation: A search method for belief-network inference under limited resources. *International Journal of Approximate Reasoning* 15, 49-76, 1996.

[16] Pearl, J., Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence* 32, 247-257, 1987.

[17] Pearl, J., *Probabilistic Reasoning in Intelligent Systems.* Morgan-Kauffman (San Mateo), 1988.

[18] Rubinstein, R.Y., *Simulation and the Monte Carlo Method.* Wiley (New York), 1981.

[19] Shachter, R.D., Evaluating influence diagrams. *Operations Research* 34, 871-882, 1986.

[20] Shachter, R.D., Probabilistic inference and influence diagrams. *Operations Research* 36, 589-605, 1988.

[21] Shachter, R.D., D'Ambrosio, B., Del Favero, B.A., Symbolic probabilistic inference in belief networks. Proceedings of the AAAI'90 Conference, Vol.1, 126-131, 1990.

[22] Shachter, R.D., Peot, M.A., Simulation approaches to general probabilistic inference on belief networks. In: *Uncertainty in Artificial Intelligence, 5* (M. Henrion, R.D. Shachter, L.N. Kanal, J.F. Lemmer, eds.) North Holland (Amsterdam) 221-231, 1990.

[23] Shachter, R.D., Andersen, S.K., Szolovits, P., The equivalence of exact methods for probabilistic inference on belief networks. Submitted to *Artificial Intelligence*, 1991.

[24] Shafer, G., Shenoy, P.P., Probability propagation. *Annals of Mathematical and Artificial Intelligence* 2, 327-351, 1990.

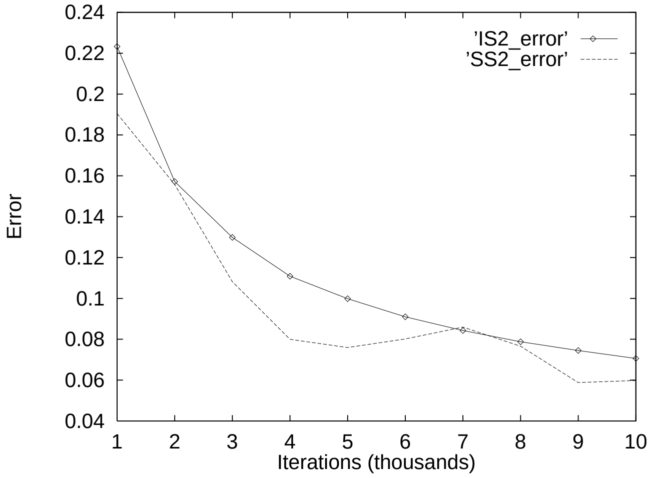

Figure 4: Experiment 1, IS2 vs SS2.

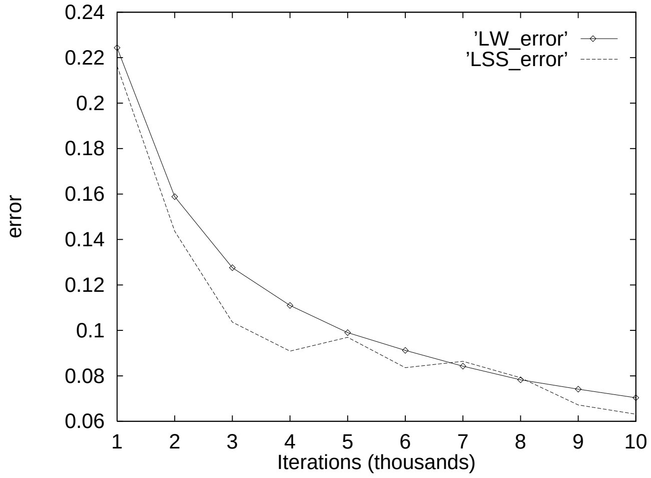

Figure 5: Experiment 1, LW vs LSS.

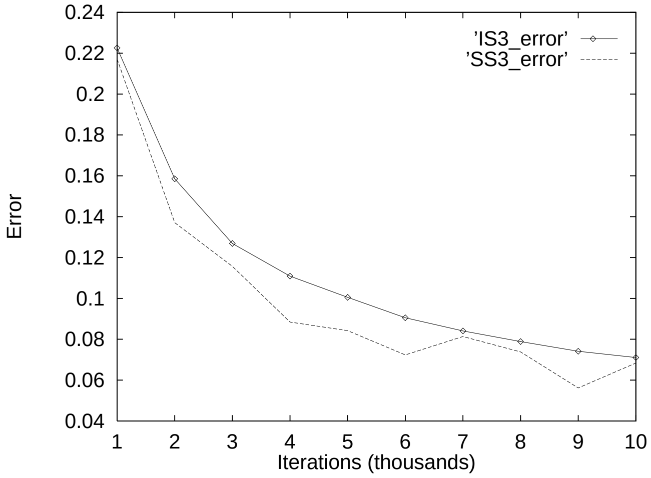

Figure 6: Experiment 1, IS3 vs SS3.

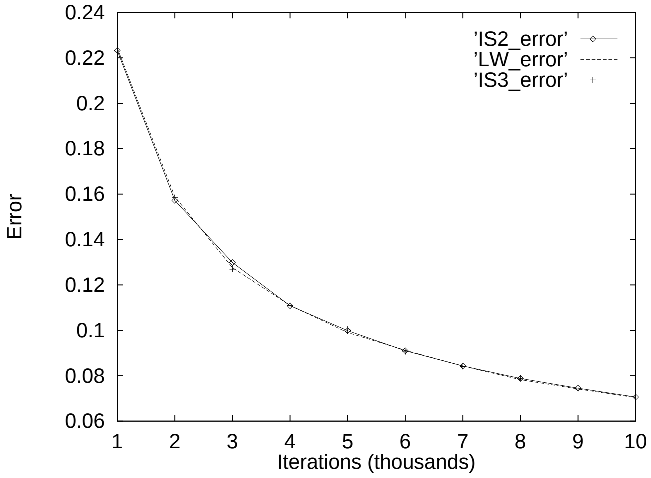

Figure 7: Experiment 1, Importance Sampling.

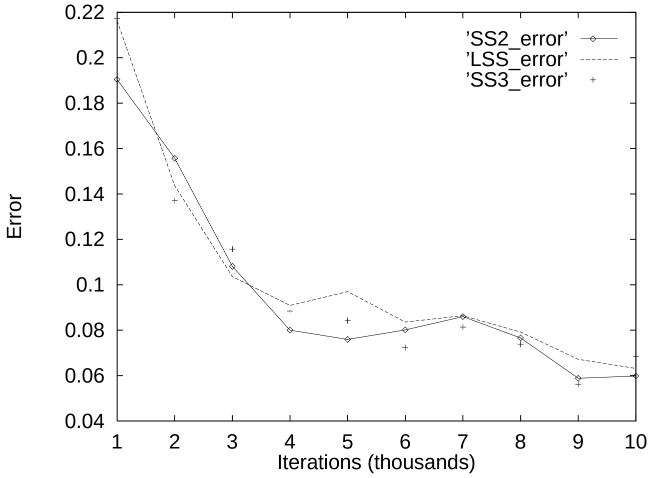

Figure 8: Experiment 1, Stratified Sampling.

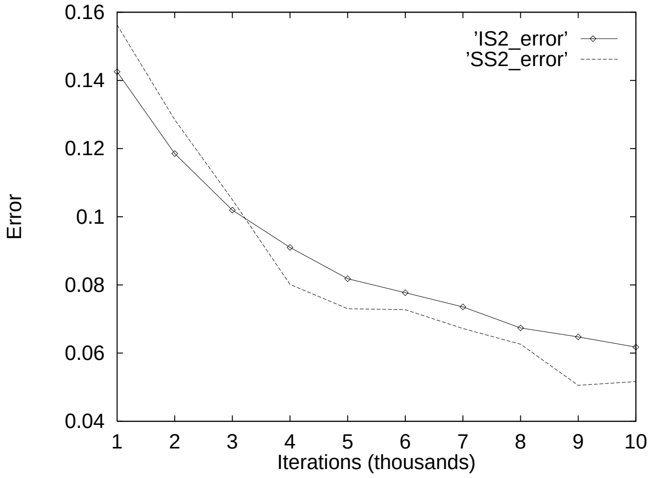

Figure 9: Experiment 2, IS2 vs SS2.

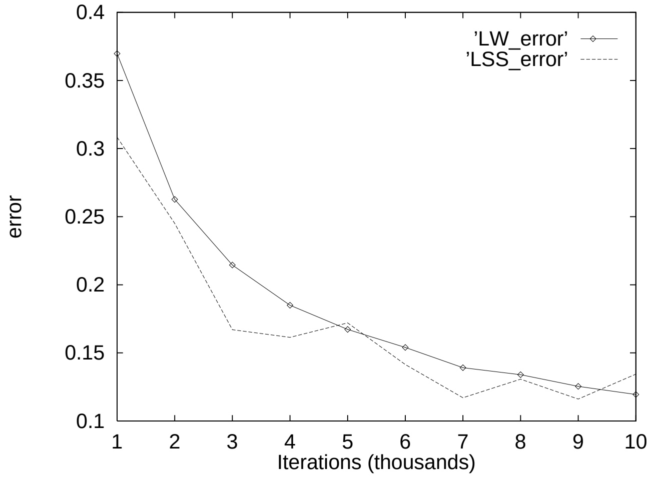

Figure 10: Experiment 2, LW vs LSS.

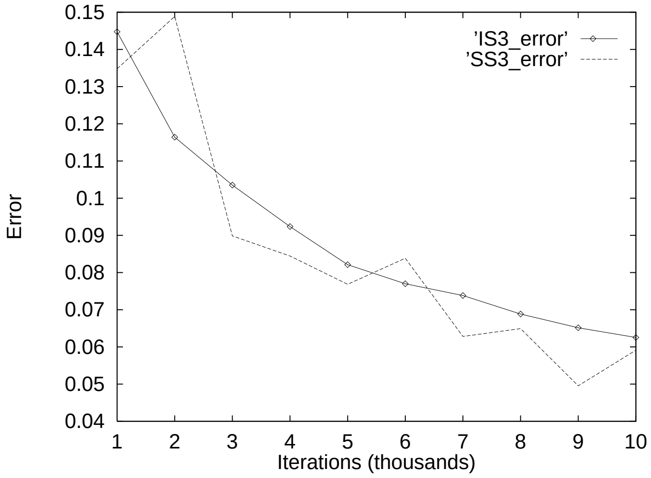

Figure 11: Experiment 2, IS3 vs SS3.

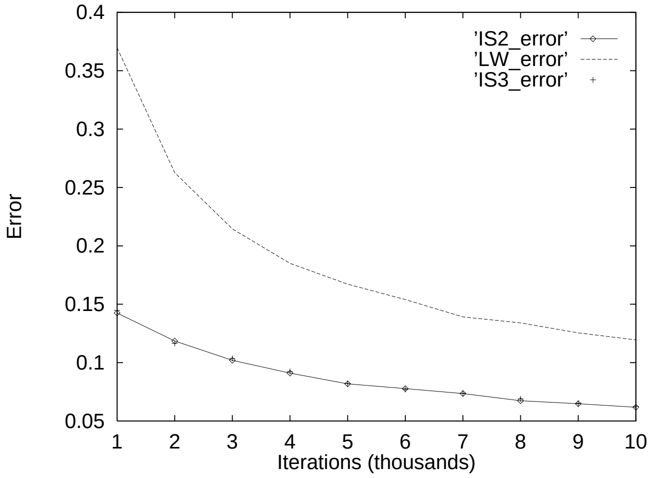

Figure 12: Experiment 2, Importance Sampling.

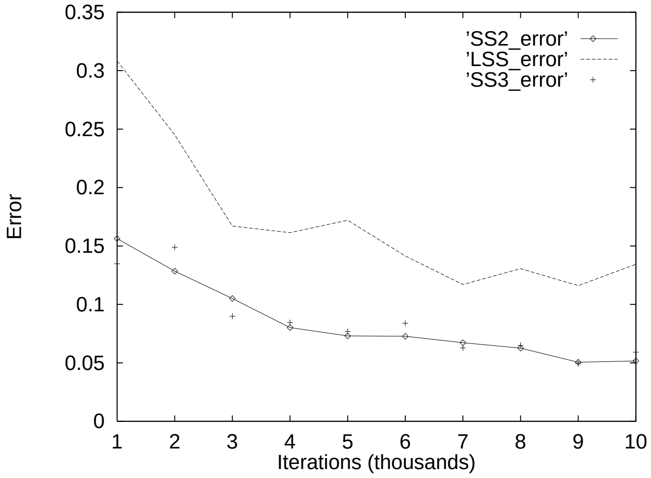

Figure 13: Experiment 2, Stratified Sampling.

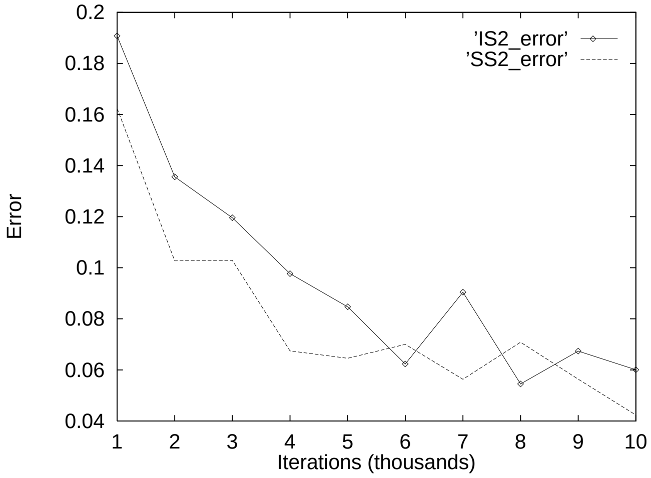

Figure 14: Experiment 3, IS2 vs SS2.

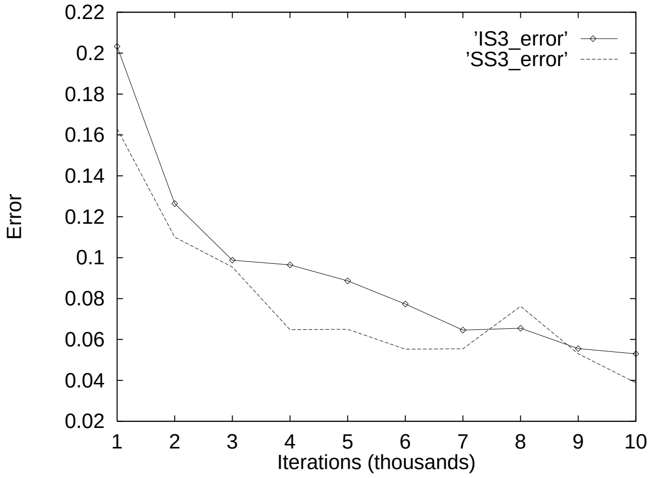

Figure 15: Experiment 3, IS3 vs SS3.

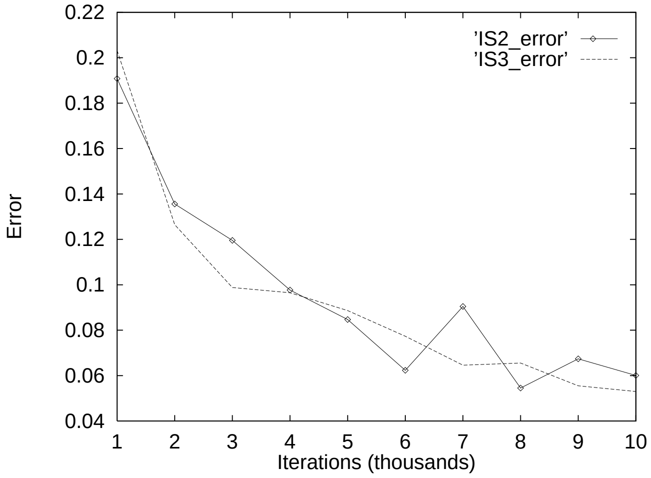

Figure 16: Experiment 3, Importance Sampling.

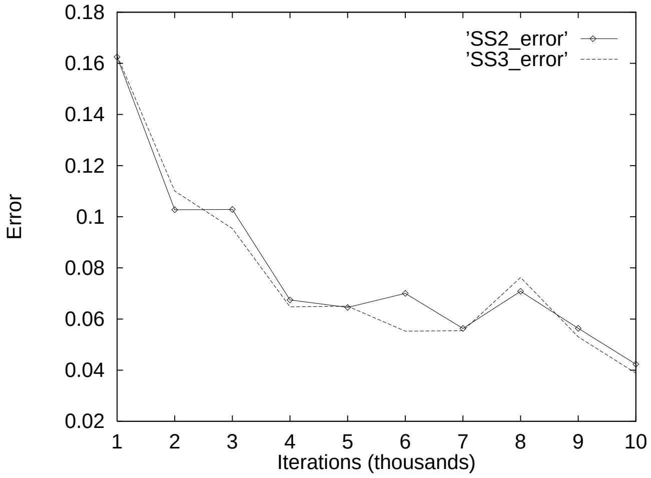

Figure 17: Experiment 3, Stratified Sampling.