Deep Bucket Elimination

Rina Dechter, Yasaman Razeghi, Yadong Lu, Pierre Baldi, Kalev Kask
University of California Irvine
{dechter, yrazeghi, yadongl1, pfbaldi, kkask}@ics.uci.edu

Abstract

Bucket Elimination (BE) is a universal inference scheme that can solve most tasks over probabilistic and deterministic graphical models exactly. *BE* works by eliminating (marginalizing) variables one by one, and generating an intermediate bucket function for each one of them sequentially. However, it often requires exponentially high levels of memory (in the induced-width) preventing its execution. In the spirit of exploiting Deep Learning for inference tasks, in this paper, we will use neural networks to approximate the bucket function computation. In our scheme training is carried out for each problem and for each of its variables, independently. The resulting *Deep Bucket Elimination (DBE)* algorithm is developed for computing the partition function. We provide a proof-of-concept empirically using instances from several different benchmarks, showing that *DBE* can be more effective than current state-of-the-art approaches (e.g. the mini-bucket schemes) especially when the problems are sufficiently hard.

Introduction

Probabilistic graphical models, including Bayesian networks and Markov random fields, provide a framework for information representation and reasoning (Pearl 1988; Spiegelhalter and Lauritzen 1990; Frey 1998; Jordan 1998; Koller and Friedman 2009; Darwiche 2009; Dechter 2013a). *Bucket Elimination* (Dechter 1999) is a universal exact algorithms for probabilistic inference. This is a variable elimination algorithm that can answer a variety of queries, ranging from constraint satisfaction to pure combinatorial optimization (e.g., Most Probable Explanation (MPE/MAP)) and weighted counting (Partition Function, Probability of Evidence, Solution Counting). Even the most challenging mixed tasks, involving both optimization and summation, such as computing the Marginal Map or the Maximum Expected Utility over an influence diagram can be addressed with *BE* (Dechter 2013a). The *BE* algorithms take advantage of the structure of the model's graph. Most bucket-elimination algorithms, however, are time and space exponential in the induced-width of the underlying dependency primal graph of the model. So when the induced-width is too high the algorithm cannot be executed. In this work, we propose to address this fundamental problem by using deep learning to approximate the *BE* functions.

To better see the problem, note that the central operation of *BE* is processing the bucket functions of each variable one at a time, in sequence. Processing a bucket involves combining all its functions by product (or by other operators) and then *eliminating* the bucket's variable by a summation or an optimization operator to generate the output bucket's function λ (also called a message). The arguments (called scope) of λ is the set of all the bucket's variable excluding itself. The bucket's function λ is then placed in its parent bucket which is associated with the variable closest in the ordering. The bucket's function is normally assumed to be expressed as a table over all the configurations of its scope. However, if the table is too large to fit memory, as it is exponential in the bucket's scope, where the maximum scope is known as the induced-width, computing and memorizing the bucket's function is not feasible.

Since *BE* cannot be executed unless the induced-width is bounded, significant research was carried out over the years exploring bounding schemes and Monte-Carlo methods (Dechter and Rish 2003; Liu and Ihler 2013, 2011, 2012). Methods based on compact representation such as decision diagrams and exploiting determinism were also explored (Dechter and Larkin 2001; Larkin and Dechter 2003; Mateescu, Dechter, and Marinescu 2008; Chavira and Darwiche 2007; Gogate and Domingos 2013).

The approach we take here is based on *Deep Learning (DL)* (Goodfellow, Bengio, and Courville 2016; Baldi 2020), leveraging the well known universal approximation properties of neural networks (NN) (Hornik, Stinchcombe, and White 1989; Cybenko 1989; Baldi 2020) to approximate the intermediate bucket functions generated by *BE* algorithms. To prove the concept, we focus on the sum-product tasks, yet our scheme is immediately applicable to optimization and mixed, max-sum queries. In probabilistic graphical models, the sum-product problem, which includes the Partition Function and the Probability of Evidence as special cases, has many applications in areas such as protein side chain prediction, genetic linkage analysis, and scheduling (Fishelson, Dovgolevsky, and Geiger 2005; G. Verfaillie and Schiex 1996; Sontag et al. 2008).

In summary, our *Deep Bucket Elimination (DBE)* scheme addresses the memory bottleneck of model-based inference

algorithms such as bucket elimination by training NNs to approximate the generated bucket-functions. Our empirical results show that *DBE* can generate significantly more accurate results compared with weighted mini-bucket, a state-of-the-art scheme, on hard instances from several benchmarks. However, *DBE* is not yet competitive time-wise since it requires training a sequence of NNs for each problem instance. Overall, *DBE* is suitable for hard enough problems, which cannot be addressed satisfactorily with current methods. Since BE's relaxations such as the mini-bucket-based schemes are the primary way of generating heuristics for subsequent search. Therefore, *DBE* has the potential for yielding superior heuristics (albeit, not necessarily upper bounds). Therefore, at this initial exploration stage, we focus more on *DBE's* accuracy, leaving speed optimization issues for followup studies.

Related Work

As note, approximating and bounding the *Bucket Elimination* algorithm has been carried out extensively over the years for all probabilistic queries. Well known is the *Mini-Bucket Elimination* scheme (Dechter and Rish 2003) and its variants, such as *Weighted Mini-Bucket (WMB)*, augmented with message-passing cost-shifting (Liu and Ihler 2013, 2011). Those schemes also extend into iterative versions such as generalized belief propagation (IBP, IJGP) (Mateescu et al. 2010; Liu and Ihler 2012; J. S. Yedidia and Weiss 2005; M. J. Wainwright and Willskey 2005). Some recent work for mixed queries appears in (Lee, Ihler, and Dechter 2018; Lee et al. 2019; Ping, Liu, and Ihler 2015). Anytime schemes that augment the *mini-bucket* scheme with sampling can enhance accuracy as more time is available (Liu, Fisher III, and Ihler 2015; Gogate and Dechter 2011). Here we will compare against weighted mini-bucket only, aiming to compare more broadly in our future work.

Other work aiming to harness neural network technology and the primary inspiration of our work, is Deep Reinforcement Learning (DRL) (Mnih et al. 2015, 2013; F. Agostinelli and Baldi 2018; Agostinelli et al. 2019). In reinforcement learning the value function is learned from temporal trajectories (samples) created from real executions or simulated ones. The value function can also be obtained by a variable elimination scheme over an underlying Markov Decision Process (MDP) (Sutton and Barto 2018; François-Lavet et al. 2018). Recent efforts in Tractable learning and reasoning, e.g., using cutset networks (Rahman, Jin, and Gogate 2019) are also related, but work within a search framework. Our approach may seem related to Graph Neural Networks (Baldi 2020; Z and Savelsbergh 1999; Scarselli et al. 2009; Gilmer et al. 2017; Yoon et al. 2018; Heess, Tarlow, and Winn 2013a) in that it is a message-passing algorithm exploring the underlying graph structure of the problem. It clearly shares the aim of bringing neural network into reasoning tasks. However our scheme differ significantly from those based on GNN. We explore learning within each problem instance only and there is currently no attempt at learning across problem instances. In particular We do not have augmented variables or hidden structures. Our algorithm builds ambitiously, on an exact scheme (BE) rather then on an approximate message-passing scheme (Heess, Tarlow, and Winn 2013b; Yoon et al. 2019). All our NN training is internal to the instance. It is not end-to-end but rather local to each bucket message and no learning across instances is aimed for. Another related work is (Silvestri, Lombardi, and Milano 2020), in which they showed some initial empirical results on using NNs in solving constraint satisfaction problems.

Background

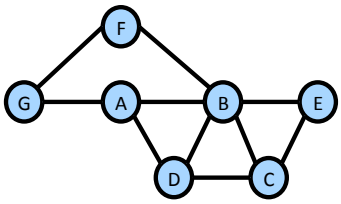
A graphical model, such as a Bayesian or a Markov network (Pearl 1988; A 2009; Dechter 2013b) can be defined by a 3-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, where $\mathbf{X} = \{X_i : i \in V, V = \{1, ..., n\}\}$ is a set of n variables indexed by V and $\mathbf{D} = \{D_i : i \in V\}$ is the set of finite domains for each X_i (i.e. each X_i can only assume values in D_i, and each D_i is finite). Each function $f_\alpha \in \mathbf{F}$ is defined over a subset of the variables called its scope, X_α, where $\alpha \subseteq V$ are the indices of variables in its scope and D_α denotes the Cartesian product of their domains, so that $f_\alpha : D_\alpha \rightarrow R \geq 0$. The **primal graph** of a graphical model associates each variable with a node. An edge between node i and node j is created if and only if there is a function containing X_i and X_j in its scope. Graphical models can be used to represent a global function, often a probability distribution, defined by $Pr(X) \propto \prod_\alpha f_\alpha(X_\alpha)$. An important task is to compute the normalizing constant, also known as the partition function $Z = \sum_X \prod_\alpha f_\alpha(X_\alpha)$.

Algorithm 1: (Deep) Bucket Elimination (DBE)

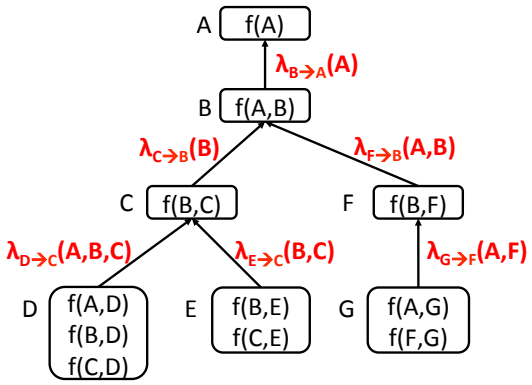
Input: Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, Ordering $d = X_1, ..., X_n$, i-bound i, ϵ, #samples ns
Output: the partition function constant and messages $\lambda_{q \rightarrow p}$

1 **foreach** p *from* n *to* 1 **do**
2 (Initialize buckets) put all unplaced functions mentioning X_p in B_p.
3 **foreach** p *from* n *to* 1 **do**
4 Let X_a be closest ancestor variable in d of X_p that is in bucket B_p
5 Generate the bucket function: $\lambda_{p \rightarrow a} \leftarrow \sum_{X_p} \prod_{f_\alpha \in B_p} f_\alpha$
6 If $width(X_P) \leq i$ then $\mu_{\Phi, p \rightarrow a} \leftarrow \lambda_{p \rightarrow a}$,
7 else
8 $\mu_{\Phi, p \rightarrow a} \leftarrow$ approximate-NN ($\lambda_{p \rightarrow a}$, ϵ, ns)
9 Put $\mu_{\Phi, p \rightarrow a}$ in B_a
10 **return** *All* μ_Φ*-messages generated (product of messages* B_1 *is the* $p(e)$*)*

Bucket Elimination Given a variable ordering, *BE* (presented in Algorithm 1, when excluding steps 6,7 and 8) processes variables one by one with respect to a given ordering. For the next variable X_i, it collects all the functions not-yet-processed having X_i in their scopes into a bucket of X_i



(a) A primal graph.



(b) Bucket elimination example

Figure 1: (a) A primal graph. of a graphical model with 7 variables. (b) Illustration of bucket elimination.

denoted B_i. This includes the original functions in the models as well as the messages created by processing previous variables. We do not distinguish between the different functions as shown in step 5. It then marginalizes X_i out from the product of functions in B_i generating a new, so called, *bucket function* or *bucket function*, denoted $\lambda_{i\to\pi_i}$, or λ_i for short.

$$\lambda_{i\to\pi_i} = \sum_{X_i} \prod_{f_\alpha \in B_i} f_\alpha$$

where X_{π_i} is the closest variable in the scope of this new function along the ordering (constants are placed in B_1). The bucket's function is placed in the bucket of X_{π_i}, for later processing. Once all the variables are processed, *BE* outputs the exact value of Z by taking the product of all the constant present in the bucket of the first variable. **Figure 1a** shows a primal graph of a graphical model with variables indexed from A to G with binary functions over pairs of variables connected by an edge. In this particular example $F = \{f(A), f(A,B), f(A,D), f(A,G), f(B,C), f(B,D), f(B,E), f(B,F), f(C,D), f(C,E), f(F,G)\}$.

Bucket-Elimination can be viewed as a message-passing algorithm along its *bucket-tree*. The nodes of the tree are the different buckets. Each bucket of a variable contains a set of the model's functions depending on the given order of processing (see Algorithm 1). There is an arc from the B_j to a parent bucket B_i, if the function created at bucket B_j is placed in bucket B_i. We illustrate *BE* message flow on our example problem in **Figure 1b**.

Complexity Both the time and space complexity of *BE* are exponential in the **induced-width** of the primal graph along the ordering, which is the size of the largest number of variables, (minus 1) in the scope of any message computed. The induced-width can be computed as a graph parameter based on the ordered primal graph (Dechter 2013b). Clearly, *BE* becomes impractical if the induced-width is large. We denote by scope(f) the set of arguments of function f. In particular $scope(B)$ is the set of variables in bucket B.

Deep Bucket Elimination

Algorithm 1 presents *DBE* which is identical to *BE*, except that when the induced-width is beyond a given i-bound, it approximates the bucket's function by training a neural network as described in its steps 6,7 and 8. As before central operation of *BE* is processing the bucket of each variable one at the time. but, when the bucket's functions are too large, *BE* cannot be executed. To overcome this limitation, we estimate the bucket's function by training a neural network having a manageable size, aiming toward achieving an error bounded by a given . The error function used is the average mean-square error. For example, in Figure 1, instead of sending an exact function from the bucket of D, $\lambda_{D\to C}(A,B,C)$, we can send a compact approximation $\mu_{\Phi,D\to C}(A,B,C)$.

Deep Bucket Elimination *DBE's*, approximation is carried out by training a neural network procedure *approximate-NN* presented in Algorithm 2. Assuming a given neural network structure and a target function λ, the training scheme first generates a given number of samples from the function and then trains the neural network until a provided error bound is obtained or until a cap on the number of training iterations (#epochs) is reached.

In principle, neural networks can approximate any reasonable function (Hornik, Stinchcombe, and White 1989; Cybenko 1989; Baldi 2020). So, the main empirical question is how to fit the NN's architecture and size to estimate a potential bucket-function, while maintaining a desirable error bound, and how these errors translate into a global error. We next provide additional details.

Algorithm 2: approximate-NN(λ, ϵ, ns)

Input: λ function on a set of variables X, ϵ bounds the bucket's error, $\#epochs$ a bound on the number of epochs, NN, the neural network structure, ns number of samples

Output: $\mu_\Phi(x)$, the trained neural network

1 samples ← generate-samples(λ, ns)
2 p=1
3 **while** $error_\Phi \geq \epsilon$;
4 & $p \leq$*#epochs* **do**
5 $\quad \mu_\Phi \leftarrow train(NN, p, samples)$
6 $\quad p \leftarrow p$+*increment*
7 $\quad$ *update error*
8 **return** μ_Φ *and a bound on the error*

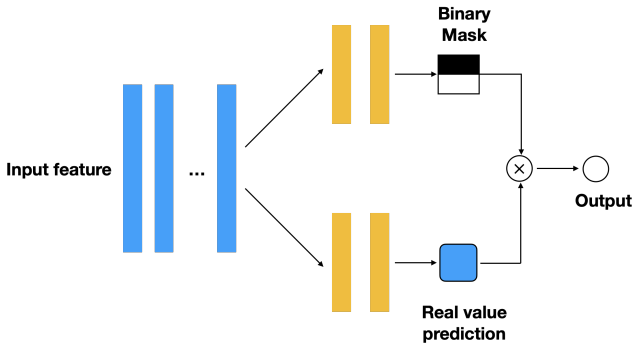


Figure 2: The structure of the MaskedNet.

The Neural Network Architecture We use two types of NNs treating differently problem instances with and without determinism (i.e. having 0 within a bucket function). For instances without determinism, we use a feedforward neural network with fully connected layers and ReLU activation function for each layer. The number of layers and the number of units in each layer can be tuned according to the complexity of the instance. For instances with determinism, we propose the MaskedNet, which first predicts whether the output is deterministic, namely whether it is zero, and if not, predicts the actual value. The structure of the MaskedNet is described in Figure 2. The input is sent to a couple of fully connected layers to obtain a feature vector which summarizes the high level representation of the input. Then this feature vector is sent to two sister networks: 1) a network that outputs a binary mask which is responsible for determining whether the final output is zero, and 2) a network responsible for predicting the target value of the Bucket's function. The activation functions of the final layer for the first and second sub-networks are the logistic function and the softplus function, respectively. The outputs from the two sister networks are multiplied together to get the final output of the MaskedNet. In our experiments, we found that the MaskedNet effectively decreases the error of the Bucket's function approximation compared to a plain feedforward network.

Sample Generation Given a bucket B_p of X_p we generate the required #samples uniformly at random from its bucket's function to serve as our training set. The bucket's function λ_p is defined in Equation 1.

$$\lambda_p = \sum_{X_p} \prod_{f_\alpha \in B_p} f_\alpha \tag{1}$$

We first generate a random configuration from the function's domain which will be in the input layer of the network. We subsequently match it with its function's value computed as follows. Specifically, given a configuration $\bar{x}_s$ selected randomly where $S = scope(B_p)$ excluding X_p, then for each $x_p \in D_p$ we compute the product $\psi(\bar{x}_s, x_p) = \prod_{f_\alpha \in B_p} f_\alpha$ (a product of constants) and subsequently sum over the different values of x_p yielding the function value $\lambda_p(\bar{x}_s)$.

Training the Neural Network Once the samples are available, we train the NN by minimizing the mean square error of the output prediction and the target value by stochastic gradient descent. For the optimization we use the Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.001 and mini-batchsize of 256. At the end of each iteration, we compute the mean squared error between the neural network output and the target value on the training set. We stop training when we acheiev the ϵ error on the training set, or when the number of maximum number of epochs is reached. When necessary, additional hyperparameter optimization is carried out using the Sherpa software (Hertela et al. 2020).

Empirical Evaluation

Algorithms We run experiments using the *DBE* algorithm and compared its performance against the weighted mini bucket *WMB* scheme (Dechter and Rish 2003; Liu and Ihler 2013). We use the *WMB* as our baseline because this scheme also directly aims to address the same memory bottleneck of *BE*, and it is controlled by the same i-bound parameter. Higher i-bound generally leads to stronger and more accurate bounds for the mini-bucket scheme. Therefore, using the same i-bound parameter ensures that similar memory resources are used, allowing for a meaningful comparison.

Benchmarks We carried out experiments on instances selected from three well-known benchmarks from the UAI repository (Easy Grids, Hard grids, Pedigrees, and DBNs). In each benchmark we distinguish between problems that can be solved exactly, which we call "easy", and those that cannot be solved, called "hard". We also distinguish benchmarks that possess *determinism*, namely having a high proportion of zero probabilities, a feature which can impact training. We selected 6 instances from the relatively easy Grids (width 20-30), 6 from the hard Grids (1600 variables, width 55), 6 instances from the Pedigree benchmark, a benchmark with high levels of determinism, and 6 instances from the DBN benchmark, totalling 24 instances.

Performance Measure To evaluate the performance, we calculate the error: $error = |log_{10}Z^* - log_{10}\hat{Z}|$ where $\hat{Z}$ is the generated estimate of the partition function, Z, and Z^* is the exact reference. When the exact Z is not available (for hard Grid benchmark), Z^* is a surrogate to Z, which is obtained from (Kask et al. 2020). Their estimate is obtained using an advanced sampling scheme for a duration of $100 * 1hr$. Another method we used to address the lack of exact Z is to convert a solvable problem instance into a hard one for *BE*, by selecting a variable ordering having a high induced-width. We used this methodology for the Pedigree benchmark, since all Pedigree instances can be solved exactly.

Results

Our overall results are shown in the four tables of Figure 3, one for each benchmark. Each table displays the number of trained buckets (#NN) and the error obtained by both

i-bound=10					DBE		WMB
Id	name	d	#v	w	#NN	error	error
1	grid1010f10w	2	100	21	31	**0.591**	32.004
2	grid1010f10	2	100	13	8	**0.169**	1.582
3	grid2020f2	2	400	27	114	**1.15**	11.24
4	grid2020f10	2	400	27	114	**13.38**	80.86
5	grid2020f5	2	400	27	114	**2.93**	39.435
6	grid2020f15	2	400	27	114	**1.81**	122.91

(a) **Grid, Easy, Without Determinism**

i-bound=20					DBE		WMB
Id	name	d	#v	w	#NN	error	error
1	grid4040f10	2	1600	55	308	**8.45**	215.45
2	grid4040f5	2	1600	55	308	**20.17**	84.92
3	grid4040f2	2	1600	55	308	**24.92**	25.24
4	grid4040f2w	2	1600	55	376	**21.33**	32
5	grid4040f15	2	1600	55	308	**72.19**	338.2
6	grid4040f15w	2	1600	55	376	**207.73**	657.03

(b) ***Grid, Hard, Without Determinism**

i-bound=20					DBE		WMB
Id	name	d	#v	w	#NN	error	error
1	pedigree40	7	842	30	92	**5.9017**	6.7977
2	pedigree41	5	885	32	92	**1.1362**	4.1497
3	pedigree7	4	867	34	108	**3.5788**	6.0012
4	pedigree34	5	922	33	106	**2.1728**	7.0762
5	pedigree31	5	1006	30	85	**8.0972**	12.3603
6	pedigree19	5	693	28	43	**1.4428**	2.5809

(c) **Pedigree, Hard, With Determinism**

i-bound=20					DBE		WMB
Id	name	d	#v	w	#NN	error	error
1	rbm20	2	40	21	20	0.0376	**0.0007**
2	rbm21	2	42	22	22	**0.1787**	6.3913
3	rbm22	2	44	23	24	**0.5726**	8.6549
4	rbm-ferro20	2	40	21	20	0.315	**0.005**
5	rbm-ferro21	2	42	22	22	2.803	**1.984**
6	rbm-ferro22	2	44	23	24	0.526	**0.517**

(d) **DBN, Easy, Without Determinism**

Figure 3: Results on performance of *DBE* against *WMB*. d:domain size, $\#v$:variable numbers, w:induced width, $\#NN$: number of NNs, $error$: L1 error for referenced and estimated $log(Z)$. *Note: Here, referenced Z is approximated by (Kask et al. 2020)

schemes (*DBE* and *WMB*). In all the experiments we used 5×10^5 samples for training the NNs with an error bound of $\epsilon = 10^{-6}$. As noted, at this preliminary stage of our work we were not concerned with the time performance but aimed primarily at accurate results. Tuning $\#$samples, is left for future work. We also manually tuned the number of epochs $\#epochs$ to the training error (see Algorithm 2). Namely, if the training error obtained was not acceptable (far from ϵ), we increased $\#epochs$. Additional details regarding the experiments and the results are provided in the Supplementary Material section.

Grids The results for the Grid benchmarks are shown in Figures 3b (hard) and 3a (easy), respectively. For the relatively easy problems we used a lower i-bound of 10 to facilitate the training of a relatively large number of buckets. As expected, when an instance has a low induced-width only a small number of buckets are trained (e.g. Id 2) and both schemes obtain high accuracy. As the induced-width is increased, more buckets are trained yet *DBE* still obtains high accuracy. In all cases, *DBE* has significantly better performance than *WMB*. On the hard instances, (Figure 3b) we used the maximum possible i-bound of 20. We observe that *DBE* can achieve a far lower error compared with *WMB*. For all the Grids we used feedforward neural networks with 2 hidden layers of 100 nodes each, and with number of epochs bounded by 10, which was sufficient to obtain good results.

Pedigree Pedigree results are presented in Table 3c. Since Pedigrees can be solved exactly with good variable orderings, alternate variable ordering were used to create hard Pedigree problems for our experiments. As alternate orderings do not change Z, the exact Z was available for reference. Here also we see that *DBE* can achieve low error, which is far smaller than the error achievable by *WMB* with the same i-bound. As expected, the number of trained buckets vary with the induced-width. The Pedigree instances include determinism, hence, we used the MaskedNet architecture for the NNs on this benchmark as we described earlier. We used networks having 3 hidden layers each containing 100 nodes for the MaskedNet (One for the blue and one for each of the yellow layers in Figure 2). The $\#epochs$ was bounded at 20 for all instances except for ids 2 and 3 where we used 40 and 80 bounds, respectively.

DBN The results for the DBN benchmark are shown in Figure 3d. All these instances have an exact solution, yet they are almost intractable memory-wise having widths in the range 20-23. Here too *DBE* can achieve a high accuracy, sometime significantly better than the *WMB* algorithm (instances 2 and 3) that appear harder in this set. *DBE* achieves low error (instances 1,4 and 6), for which *WMB* is almost exact. We used feedforward architecture having 2 or 3 number of hidden layers based on the complexity of the instances and 100 number of nodes in each layer. The $\#epochs = 10$ for instances ids $\{1, 2, 3\}$, $\#epochs = 20$ for instance id 5, and is is 60 and 100 for instances 4 and 6, respectively.

The Impact of the i-bound Normally, the highest i-bound possible is 20, depending also on the domain size. For the *WMB* schema, both theory and practice show that higher i-bound often yield more accurate approximations (Dechter and Rish 2003). In our experiments we initially used several

Grid					DBE i-bound 20			DBE i-bound 15		
Id	name	H	#v	w	#NN	error	T(s)	#NN	error	T(s)
1	grid2020f10	e	400	27	31	0.53	3976	69	4	10098
2	grid2020f5	e	400	27	31	2.026	4048	69	2.84	8444
3	grid4040f10	h	1600	55	308	8.45	49711	421	69.91	54133
4	grid4040f5	h	1600	55	308	20.17	39608	421	37.79	53037

Figure 4: The impact of the i-bound, H:hardness, e:easy, h:hard, $\#v$:number of variables, w:induced width, $\#NN$: number of NNs, $error$: L1 error for referenced and estimated $log(Z)$, T: algorithm running time.

i-bounds to explore the impact of this parameter and we immediately observed that with higher i-bound *DBE* achieves higher accuracy with less time. To illustrate this point, we show results on four grid instances running with i-bounds of $i = 20$ and $i = 15$. The table in Figure 4 depicts the error and the time for both algorithms. We clearly see that *DBE*'s accuracy and time is significantly better when the i-bound is higher and that the number of buckets that need to be trained is reduced. With fewer trained buckets, we expect higher accuracy (as more functions are exact) and lower times (as we have less functions to train).

Summary of Experiments Our empirical evaluation shows that *DBE* is effective at generating accurate estimates of the partition functions, and is superior to the *WMB* baseline, especially when problem instances are hard.

We acknowledge that *DBE* is inherently time consuming, far more than *WMB* since it involves training multiple (e.g., hundreds) NNs. Often *WMB* takes seconds, or at most minutes, when *DBE* may take hours. On the other hand, as we noted, *WMB* is not able to improve its performance, even with more time, whereas *DBE* is able to do so, with room for improving its speed in future studies.

Conclusion and Future Work

The paper presents *Deep Bucket Elimination (DBE)*, a new algorithm that harnesses neural network technology to advance the universal Bucket Elimination algorithm. We provide an initial empirical evaluation showing very good performance on challenging instances from three benchmarks. In particular, its superior performance against the *weighted mini-bucket* scheme, is clearly established.

We should emphasize that the proposed *DBE* scheme is novel in the way it incorporates NNs into traditional graphical models algorithms. Although the scheme is still in its infancy, it opens new doors for exploring these methodologies. In particular, it opens up reasoning algorithms into a new modus operandi that tolerates significant pre-processing and interaction with the user, for the sake of achieving better solutions to challenging reasoning tasks.

Future work We chose to compare against *WMB* since it is a widely known scheme that shares the same structure and some of the same properties as *DBE*. The *WMB* will not be able to improve its performance even given more time. That being said, other schemes developed in recent years are anytime and were shown to be more powerful (Liu, Fisher III, and Ihler 2015; Gogate and Dechter 2011). Some of these schemes augment the *mini-bucket* approach with search or sampling facilitating more accurate solutions or tighter bounds with more time. Consequently, the primary focus of our future work will be on studying *DBE*'s potential against advanced anytime schemes (*WMB-IS* or *Dynamic-IS* (Liu, Fisher III, and Ihler 2015; Gogate and Dechter 2011)).

We will also explore speeding up the training in *DBE*. Many design choices we made were addressed to provide a proof-of-concept regarding accuracy while deferring efficiency issues to future work. For example, we use the same network architecture and the same number of samples across all the instances of a benchmark and across all their buckets, ignoring their function complexity (e.g., number of arguments). We will also explore ideas from transfer learning (Pan and Yang 2010; Weiss, Khoshgoftaar, and Wang 2016) techniques; retraining a network to approximate a specific bucket function, and then fine-tuning the same network to approximate the others. In addition we will explore using different NN architectures and different sample size, depending on each bucket function complexity and width.

References

A, D. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence* 1(8): 356–363.

Baldi, P. 2020. *Deep Learning in Science: Theory, Algorithms, and Applications*. Cambridge, UK: Cambridge University Press. In press.

Chavira, M.; and Darwiche, A. 2007. Compiling Bayesian Networks Using Variable Elimination. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2443–2449.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)* 2(4): 303–314.

Darwiche, A. 2009. *Modeling and reasoning with Bayesian networks*. Cambridge university press.

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

Dechter, R. 2013a. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 7(3): 1–191.

Dechter, R. 2013b. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. doi:10.2200/

S00529ED1V01Y201308AIM023. URL http://dx.doi.org/10.2200/S00529ED1V01Y201308AIM023.

Dechter, R.; and Larkin, D. 2001. Hybrid Processing of Beliefs and Constraints. In Breese, J. S.; and Koller, D., eds., *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001*, 112–119. Morgan Kaufmann. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1\&smnu=2\&article_id=90\&proceeding_id=17.

Dechter, R.; and Rish, I. 2003. Mini-buckets: A General Scheme for Bounded Inference. *Journal of the ACM (JACM)* 50(2): 107–153.

F. Agostinelli, G. Hocquet, S. S.; and Baldi, P. 2018. From Reinforcement Learning to Deep Reinforcement Learning: An Overview. In Muchnik, I., ed., *Key Ideas in Learning Theory from Inception to Current State: Emmanuel Braverman's Legacy. Volume in the Springer series: LNCS State-of-the-Art Surveys*, 269–297. Springer.

Fishelson, M.; Dovgolevsky, N.; and Geiger, D. 2005. Maximum likelihood haplotyping for general pedigrees. *Human Heredity* .

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; and Pineau, J. 2018. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560* .

Frey, B. 1998. Cambridge, MA: MIT Press.

G. Verfaillie, M. L.; and Schiex, T. 1996. Russian doll search for solving constraint optimization problems.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. URL https://arxiv.org/abs/1704.01212.

Gogate, V.; and Dechter, R. 2011. SampleSearch: Importance sampling in presence of determinism. *Artif. Intell.* 175(2): 694–729.

Gogate, V.; and Domingos, P. M. 2013. Structured Message Passing. In Nicholson, A.; and Smyth, P., eds., *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. AUAI Press. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1\&smnu=2\&article_id=2386\&proceeding_id=29.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Heess, N.; Tarlow, D.; and Winn, J. 2013a. Learning to Pass Expectation Propagation Messages. In Burges, C. J. C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems*, volume 26, 3219–3227. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2013/file/1714726c817af50457d810aae9d27a2e-Paper.pdf.

Heess, N.; Tarlow, D.; and Winn, J. M. 2013b. Learning to Pass Expectation Propagation Messages. In Burges, C. J. C.; Bottou, L.; Ghahramani, Z.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, 3219–3227. URL http://papers.nips.cc/paper/5070-learning-to-pass-expectation-propagation-messages.

Hertela, L.; Collado, J.; Sadowskic, P.; Ott, J.; and Baldi, P. 2020. Sherpa: Robust Hyperparameter Optimization for Machine Learning. *SoftwareX* Submitted. Also arXiv:2005.04048. Software available at: https://github.com/sherpa-ai/sherpa.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5): 359–366. ISSN 0893-6080. doi:10.1016/0893-6080(89)90020-8. URL http://dx.doi.org/10.1016/0893-6080(89)90020-8.

J. S. Yedidia, W. F.; and Weiss, Y. 2005. Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms. *IEEE Transaction on Information Theory* 2282–2312.

Jordan, M. I. 1998. *Learning in graphical models*, volume 89. Springer Science & Business Media.

Kask, K.; Pezeshki, B.; Broka, F.; Ihler, A.; and Dechter, R. 2020. Scaling Up AND/OR Abstraction Sampling .

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. URL http://arxiv.org/abs/1412.6980. Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Koller, D.; and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press. ISBN 0262013193.

Larkin, D.; and Dechter, R. 2003. Bayesian Inference in the Presence of Determinism. In Bishop, C. M.; and Frey, B. J., eds., *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, AISTATS 2003, Key West, Florida, USA, January 3-6, 2003*. Society for Artificial Intelligence and Statistics.

Lee, J.; Ihler, A.; and Dechter, R. 2018. Join Graph Decomposition Bounds for Influence Diagrams. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1053–1062.

Lee, J.; Marinescu, R.; Ihler, A.; and Dechter, R. 2019. A Weighted Mini-Bucket Bound for Solving Influence Diagrams. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Liu, Q.; Fisher III, J. W.; and Ihler, A. T. 2015. Probabilistic Variational Bounds for Graphical Models. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*, 1432–1440. Montreal, Canada: Curran Associates, Inc.

Liu, Q.; and Ihler, A. 2012. Belief Propagation for Structured Decision Making. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, 523–532.

Liu, Q.; and Ihler, A. T. 2011. Bounding the Partition Function using Holder's Inequality. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, 849–856.

Liu, Q.; and Ihler, A. T. 2013. Variational Algorithms for Marginal MAP. *CoRR* abs/1302.6584.

M. J. Wainwright, T. J.; and Willskey, A. S. 2005. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory* 2313–2335.

Mateescu, R.; Dechter, R.; and Marinescu, R. 2008. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. *J. Artif. Intell. Res.* 33: 465–519.

Mateescu, R.; Kask, K.; Gogate, V.; and Dechter, R. 2010. Join-Graph Propagation Algorithms. *J. Artif. Intell. Res. (JAIR)* 37: 279–328.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature* 518(7540): 529–533.

Pan, S.; and Yang, Q. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22(10): 1345–1359.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Ping, W.; Liu, Q.; and Ihler, A. T. 2015. Decomposition Bounds for Marginal MAP. In *Proceedings of Advances in Neural Information Processing Systems 28*, 3267–3275.

Rahman, T.; Jin, S.; and Gogate, V. 2019. Look Ma, No Latent Variables: Accurate Cutset Networks via Compilation. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, 5311–5320. PMLR. URL http://proceedings.mlr.press/v97/rahman19a.html.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Trans. Neural Networks* 20(1): 61–80. doi:10.1109/TNN.2008.2005605. URL https://doi.org/10.1109/TNN.2008.2005605.

Silvestri, M.; Lombardi, M.; and Milano, M. 2020. Injecting Domain Knowledge in Neural Networks: a Controlled Experiment on a Constrained Problem. *arXiv preprint arXiv:2002.10742* .

Sontag, D.; Meltzer, T.; Globerson, A.; Jaakkola, T.; and Weiss, Y. 2008. Tightening LP Relaxations for MAP using Message Passing. In *UAI*, 503–510.

Spiegelhalter, D. J.; and Lauritzen, S. L. 1990. Sequential updating of conditional probabilities on directed graphical structures. *Networks* 20(5): 579–605.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Weiss, K. R.; Khoshgoftaar, T.; and Wang, D. 2016. A survey of transfer learning. *Journal of Big Data* 3: 1–40.

Yoon, K.; Liao, R.; Xiong, Y.; Zhang, L.; Fetaya, E.; Urtasun, R.; Zemel, R. S.; and Pitkow, X. 2018. Inference in probabilistic graphical models by Graph Neural Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL https://openreview.net/forum?id=rkN1pF1vz.

Yoon, K.; Liao, R.; Xiong, Y.; Zhang, L.; Fetaya, E.; Urtasun, R.; Zemel, R. S.; and Pitkow, X. 2019. Inference in Probabilistic Graphical Models by Graph Neural Networks. In Matthews, M. B., ed., *53rd Asilomar Conference on Signals, Systems, and Computers, ACSCC 2019, Pacific Grove, CA, USA, November 3-6, 2019*, 868–875. IEEE. doi:10.1109/IEEECONF44664.2019.9048920. URL https://doi.org/10.1109/IEEECONF44664.2019.9048920.

Z, Gu, G. L. N.; and Savelsbergh, M. W. P. 1999. Lifted flow covers for mized 0-1 integer programs. *Mathematical Programming* 439–467.

Supplemental Material

Experiments The parameters and results for all the experiments in the paper is provided in Figures 5, 6, 7, 8

Grid Easy						DBE						WMB	
name	#v	d	w	i	exact Z	#ep	Arch	ε	#NN	est Z DBE	error DBE	est Z WMB	error WMB
grid1010f10w	100	2	21	10	333.321	10	ff-2layers	1.00E-06	31	333.912	0.591	365.325	32.004
grid1010f10	100	2	13	10	303.086			1.00E-06	8	302.917	0.169	304.668	1.582
grid2020f2	400	2	27	10	291.733			1.00E-06	114	292.883	1.15	302.973	11.24
grid2020f10	400	2	27	10	1311.98			1.00E-06	114	1298.6	13.38	1392.84	80.86
grid2020f10	400	2	27	15	1311.98			1.00E-06	69	1315.98	4	1351.56	39.58
grid2020f10	400	2	27	20	1311.98			1.00E-06	31	1311.45	0.53	1334.83	22.85
grid2020f5	400	2	27	10	665.12			1.00E-06	114	662.19	2.93	704.555	39.435
grid2020f5	400	2	27	15	665.12			1.00E-06	69	662.28	2.84	683.478	18.358
grid2020f5	400	2	27	20	665.12			1.00E-06	31	663.094	2.026	674.731	9.611
grid2020f15	400	2	27	10	1962.98			1.00E-06	114	1964.79	1.81	2085.89	122.91

Figure 5: Easy Grid benchmark experiments. $\#v$:number of variables, d: maximum domain size , w:induced width, i:ibound, $\#ep$:bound on the number of epochs, $Arch$:NN architecture ff stands for feedforward and each layer has 100 nodes , ϵ:error bound for *DBE*, $\#NN$: number of trained NNs, $estZ$: estimated Z value, $error$: L1 error for exact and estimated $log(Z)$.

Grid Hard						DBE						WMB	
name	#v	d	w	i	ref Z	#ep	Arch	ε	#NN	est Z DBE	error DBE	est Z WMB	error WMB
grid4040f10	1600	2	55	15	5490	10	ff-2layers	1.00E-06	421	5420.09	69.91	5756.31	266.31
grid4040f10	1600	2	55	20	5490			1.00E-06	308	5481.55	8.45	5705.45	215.45
grid4040f10	1600	2	55	20	5490			1.00E-05	308	5287.29	202.71	5705.45	215.45
grid4040f5	1600	2	55	15	2800			1.00E-06	421	2762.21	37.79	2910.18	110.18
grid4040f5	1600	2	55	20	2800			1.00E-06	308	2779.83	20.17	2884.92	84.92
grid4040f2	1600	2	55	20	1220			1.00E-06	308	1244.92	24.92	1245.24	25.24
grid4040f2	1600	2	55	20	1220			1.00E-05	308	1264.32	44.32	1245.24	25.24
grid4040f2w	1600	2	55	20	1231			1.00E-06	376	1252.33	21.33	1263	32
grid4040f15	1600	2	55	20	8200			1.00E-06	308	8127.81	72.19	8538.2	338.2
grid4040f15	1600	2	55	20	8200			1.00E-05	308	8116.97	83.03	8538.2	338.2
grid4040f15w	1600	2	55	20	8230			1.00E-06	376	8437.73	207.73	8887.03	657.03

Figure 6: Hard Grid benchmark experiments. $\#v$:number of variables, d: maximum domain size , w:induced width, i:ibound, $refZ$:referenced Z value computed with 100 hours of abstraction sampling , $\#ep$:bound on the number of epochs, $Arch$:NN architecture ff stands for feedforward and each layer has 100 nodes, ϵ:error bound for *DBE*, $\#NN$: number of trained NNs, $estZ$: estimated Z value, $error$: L1 error for referenced and estimated $log(Z)$.

Pedigree Hard						DBE						WMB	
name	#v	d	w	i	exact Z	Arch	#ep	ε	#NN	est Z DBE	error DBE	est Z WMB	error WMB
pedigree40	842	7	30	20	-87.88	MN-3layers	20	1.00E-06	92	-93.7817	5.9017	-81.0823	6.7977
pedigree41	885	5	32	20	-76.04		40	1.00E-06	92	-77.1762	1.1362	-71.8903	4.1497
pedigree7	867	4	34	20	-64.82		80	1.00E-06	108	-68.3988	3.5788	-58.8188	6.0012
pedigree34	922	5	33	20	-64.23		20	1.00E-06	106	-66.4028	2.1728	-57.1538	7.0762
pedigree31	1006	5	30	20	-78.52		20	1.00E-06	85	-70.4228	8.0972	-66.1597	12.3603
pedigree19	693	5	28	20	-59.02		20	1.00E-06	43	-60.4628	1.4428	-56.4391	2.5809

Figure 7: Hard Pedigree benchmark experiments. $\#v$:number of variables, d: maximum domain size , w:induced width, i:ibound, $Arch$:NN architecture MN stands for MaskedNet and each layer has 100 nodes, $\#ep$:bound on the number of epochs, ϵ:error bound for *DBE*, $\#NN$: number of trained NNs, $estZ$: estimated Z value, $error$: L1 error for exact and estimated $log(Z)$.

DBN Easy						DBE						WMB	
name	#v	d	w	i	exact Z	#ep	ε	Arch	#NN	est Z DBE	error DBE	est Z WMB	error WMB
rbm20	40	2	21	20	58.53	10	1.00E-06	ff-2layers	20	58.4924	0.0376	58.5307	0.0007
rbm21	42	2	22	20	63.15	10	1.00E-06	ff-2layers	22	62.9713	0.1787	69.5413	6.3913
rbm22	44	2	23	20	66.55	10	1.00E-06	ff-2layers	24	65.9774	0.5726	75.2049	8.6549
rbm-ferro20	40	2	21	20	151.16	60	1.00E-06	ff-2layers	20	150.845	0.315	151.155	0.005
rbm-ferro21	42	2	22	20	152.62	20	1.00E-06	ff-2layers	22	155.423	2.803	154.604	1.984
rbm-ferro22	44	2	23	20	166.11	100	1.00E-06	ff-3layers	24	165.584	0.526	166.627	0.517

Figure 8: Easy DBN benchmark experiments. $\#v$:number of variables, d: maximum domain size , w:induced width, i:ibound, $\#ep$:bound on the number of epochs, ϵ:error bound for *DBE*, $Arch$:NN architecture ff stands for feedforward and each layer has 100 nodes ,$\#NN$: number of trained NNs, $estZ$: estimated Z value, $error$: L1 error for exact and estimated $log(Z)$.